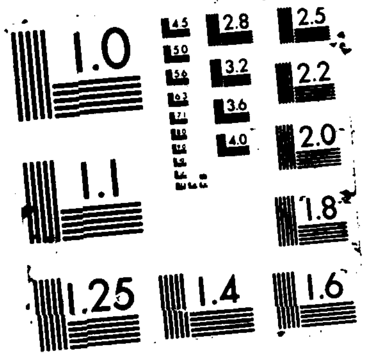
AD-A185 626 AUTOMATING RULE STRENGTHS IN EXPERT SYSTEMS(U) DUKE UNIV DURHAM NC DEPT OF COMPUTER SCIENCE M G VALTORTA MAY 87 AFOSR-TR-87-1348 $AFOSR-83-0205 1/2

UNCLASSIFIED F/G 12/9 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

~~DTIC FILE COPY~~

(2)

AD-A185 626

REPORT DOCUMENTATION PAGE

1a. REPORT	1b. RESTRICTIVE MARKINGS
2a. SECURITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE SEP 3 0 1987	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 87-1348
6a. NAME OF PERFORMING ORGANIZATION Duke University Computer Science Department	6b. OFFICE SYMBOL (If applicable)
7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research	
6c. ADDRESS (City, State and ZIP Code) Durham, North Carolina 27706	7b. ADDRESS (City, State and ZIP Code) Air Force System Command Bolling AFB, Bld 410 Washington, D.C. 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ~~Grant~~ AFOSR 83-0205	
8c. ADDRESS (City, State and ZIP Code) Bld 410 BAFB DC 20332	

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A3	

11. TITLE (Include Security Classification) Automating rule strengths in expert systems

12. PERSONAL AUTHOR(S) Marco G. Valtorta

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM 7/1/83 TO 5/31/87	1987 May	viii + 146

16. SUPPLEMENTARY NOTATION

Ph.D. dissertation

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Expert systems, reasoning with uncertainty, knowledge-based systems, rule strength determination, NP-hard problems

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Automating rule strengths in expert systems is a way to alleviate the knowledge acquisition bottleneck. It is assumed that rules are fixed, except for the values of their strengths, which are computed or adjusted from initial values given by experts.

A model of expert systems is proposed, in which rules have the form IF (P_1 & P_2 & . . . & P_n) THEN C WITH ATTENUATION a, where P_1, P_2, . . ., P_n, and C are weighted propositions, i.e., statements with a certainty factor (CF), and a, the strength of the rule, is a number between 0 and 1.

To compute rule attenuations, two problem settings are considered. In the first, an oracle is given, that can provide the CFs of the conclusions of the entire rule-based system, given any assignment of certainty factors to the premises of the entire system (complete case). In the second, a fixed set of cases is available (incomplete case). (continued on back)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
[illegible]	[illegible]	NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE

unclassified

19. (continued)

A fast algorithm for synthesis in the complete case for simple rule bases is given both for MAX and probabilistic sum.

In the incomplete case, the synthesis of attenuations is shown to be NP-Complete, evan for very shallow rule bases with only two propositions in the premise of each rule, both for MAX and probabilistic sum. The refinement of attenuations from expert-given estimates is shown to be NP-Hard, no matter how close to the correct value the estimates are and how small an improvement towards the correct value is desired.

CS-1987-15

Automating Rule Strengths in Expert Systems

Marco Valtorta

Department of Computer Science
Duke University

This Ph.D. dissertation was partially supported by Air Force Office of Scientific Research Grants AFOSR-81-0221 and AFOSR-83-0205, with my adviser D. W. Loveland as principal investigator.

ABSTRACT

Automating rule strengths in expert systems is a way to alleviate the knowledge acquisition bottleneck. It is assumed that rules are fixed, except for the values of their strengths, which are computed or adjusted from initial values given by experts.

A model of expert systems is proposed, in which rules have the form IF (P_1 & P_2 & . . . & P_n) THEN C WITH ATTENUATION a, where P_1, P_2, . . ., P_n, and C are weighted propositions, i.e., statements with a certainty factor (CF), and a, the strength of the rule, is a number between 0 and 1. Certainty factors are numbers between 0 and 1. The premise of the rule has a CF that is computed by taking the MIN of the CFs of its component conjuncts. The CF of the premise is multiplied by the strength of the rule to obtain the CF of the conclusion, C. Many rules may conclude the same proposition; their CFs are merged by using the MAX or the probabilistic addition functions.

To compute rule attenuations, two problem settings are considered. In the first, an oracle is given, that can provide the CFs of the conclusions of the entire rule-based system, given any assignment of certainty factors to the premises of the entire system (complete case). In the second, a fixed set of cases is available (incomplete case).

A fast algorithm for synthesis in the complete case for simple rule bases is given both for MAX and probabilistic sum. It is shown that, when premises are shared among rules, a well-known result in the testing of Boolean circuits implies that determination of a rule strength is NP-Complete. However, a fast algorithm is given for the determination of input attenuations.

In the incomplete case, the synthesis of attenuations is shown to be NP-Complete, even for very shallow rule bases with only two propositions in the premise of each rule, both for MAX and probabilistic sum. The refinement of attenuations from expert-given estimates is shown to be NP-Hard, no matter how close to the correct value the estimates are and how small an improvement towards the correct value is desired.

The NP-Hardness proofs are exploited to define a strict condition under which a fast algorithm for the MAX case exists and to interpret several simple iterative algorithms.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ACKNOWLEDGEMENTS

This work would not have started without the support of the Commission for Cultural Exchanges between Italy and the United States, which awarded me a Fulbright grant that paid my travel to the United States and supported me for the first nine month of my stay. Most of the work described in this dissertation was supported by grants awarded by the Air Force Office of Scientific Research (AFOSR-81-0221 and AFOSR-83-0205) to my adviser, Professor Donald W. Loveland.

Thanks to my wife, Laura, for supporting me in many ways throughout the development of the research written in this dissertation, and to my parents, who encouraged me to begin and to complete this work.

The Department of Computer Science at Duke University provided a pleasant and stimulating environment for research. Let me remember with sincere thanks some of my fellow graduate students: Mark Aulick, Mike Leuze, Mary Mace, Charles Poirier, G.B. "Gobi" Srinivasan, Happy Deas, Pam Fink, Scott Bradley, Jim Ellis, Tom Truscott, Steve Daniel, Durward Rogers, Jothy Rosenberg, John Dallen, Dave Mutchler, Mark Smotherman, Mark Rosso, Barry Koster, Joel Saltz, Roger Smith, Charlie Martin, Doug Elliott, Albert Nigrin, Victor Nicola, Amr Fahmi, and Andrew Reibman. Special thanks to Bruce Smith of UNC-Chapel Hill for the many hours of good work together on the Graduate Course Adviser.

CHAPTER ONE

INTRODUCTION

The problem in practice

One of the reasons first proposed to motivate the use of rules for expert systems is that they capture a "chunk" of an expert's knowledge in an explicit way [Davis and King, 1977]. The rules, it was proposed, could be acquired through interviews with an expert in the field of interest with concern only to the correctness of each one of the rules. The knowledge engineer's job would be to help the expert formulate the rules, extract rules from examples, and resolve lexical problems. It was suggested that the separation of rules ("knowledge") and rule interpreter ("inference engine") would allow the expert-knowledge engineer team to concentrate exclusively on the correctness of the individual rules. The inference engine would then use this knowledge to solve the problems posed to the expert system, just as the human expert uses deductive reasoning to organize fact and knowledge to solve the problems presented to him. This view was corroborated by the apparent power of production rules as a knowledge representation formalism and by advances in the practice of logic programming.

A Prolog program [Kowalski, 1979; Clocksin and Mellish, 1981] can be viewed as a deterministic expert system: rules and facts correspond to clauses and the rule interpreter corresponds to the Prolog interpreter. Prolog, at least without non-logical features such as "cut" and "var", has simple declarative semantics [Van Emden and Kowalski, 1976]. Therefore, ideally one could write a Prolog program ignoring the model of its interpreter, while concentrating on the correctness of the program's clauses, just as one needs only to concentrate on the correctness (perhaps, "adequacy" would be a better term) of the axioms describing a theorem to be solved by a theorem prover, ignoring the computational model that represents the prover itself.

The analogy between logic programming and knowledge engineering should now be apparent: in both cases, one need not bother with "how" a problem is solved (i.e., how knowledge is used, how clause are invoked), but only with "what" is stated (i.e., the conformity of rules or clauses to the expert's or programmer's view of the world). If the knowledge is correct, everything will work well. In particular, correctness of each rule is sufficient to insure correctness of the rule base. Again, there is a strong analogy between production rule systems and logic programming.

The contradiction backtracking algorithm [Shapiro, 1982; Shapiro, 1981] finds a wrong clause in a logic program that computed a wrong result, provided that an oracle exists that can decide whether any clause in the program, when instantiated, is right or wrong. Usually, the oracle is the programmer himself. In a production system, an incorrect result must also derive from an incorrect production, as argued in [Brownston et al., 1985].

Shapiro proposed to extend the contradiction backtracking algorithm to a "Prolog with uncertainties" [Shapiro, 1983]. He writes: "Assume that a logic program with uncertainties has a conclusion A whose computed certainty is judged by the expert to be too high. We can conclude that, according to the interpretation the expert has in mind, the program contains at least one false clause. Such a clause can be detected by querying the expert about the truth (or certainty) of intermediate conclusions obtained during the proof of A, as done in [Shapiro, 1982]."

I contend that this suggestion is totally impractical, because of the impossibility of an expert judging precisely what the certainty of intermediate conclusions (or, equivalently, the strengths of rules) should be. The reason for this is that even small differences in the strength of individual rules can lead to changes in the ranking of intermediate hypotheses and conclusions. In practice, therefore, the strength of a rule is correct only relatively to the strength of other rules and the correct strength of a rule depends on the strength of other rules and the mechanism used to combine them (i.e., on the inference engine).

An example of this difficulty, with a MYCIN-like system, is presented in [Brooks and Heiser, 1980]: "Suppose that the possible diagnoses for a patient are D1(0.9) and D2(0.7). [0.9 and 0.7 are the CFs of D1 and D2.] This means that the evidence in favor of D1 is stronger than that in favor of D2. Furthermore, suppose that the following two rules for medication exist.
R1: If diagnosis = D1, then medication = M1(0.6).
R2: If diagnosis = D2, then medication = M2(0.9).
When rules R1 and R2 are applied and the certainty factors are multiplied together, the certainties are
M1: 0.9*0.6 = 0.54
M2: 0.7*0.9 = 0.63.

Thus, the system would recommend medication M2 for diagnosis D1!" It seems that the designer of MYCIN anticipated this problem when he stated: "There are two kinds of medical decisions that may be involved in a [clinical decision-making] system--the determination of the patient's diagnosis or [sic] the appropriate way to treat him" [Shortliffe, 1976, p.12]. The problem described by Brooks and Heiser could be solved by splitting the system in two phases, one concerned with diagnosis, the other with treatment. Only treatment of the most likely illnesses would be addressed. (If the cost of treatment were a consideration, the expert would concentrate on treating the most likely illness only.) This two-phase decomposition would be analogous to the one used in the Graduate Course Adviser [Valtorta et al., 1984].

Still, in general, one cannot expect the expert or even the knowledge engineer to anticipate all the ways in which rule strengths and hypotheses certainties will interact in actual consultations. If the knowledge engineer could do so, there would be no need to separate rules from inference engine, and he could write a conventional program just as well. As Shapiro writes, "One reason for expert systems being composed of a rule-base and an inference mechanism is to allow the expert to effectively transfer her knowledge in a declarative form to the system, even when she is ignorant of the particular inference mechanism used."

One wants to encode correct knowledge independently of the way it will be used, but one must know how it is used in order for the knowledge to be correct. Is there a way out of this dilemma?

A methodology

The methodology that I propose to overcome the impasse described in the previous section could be summarized in the phrase knowledge refinement.

One of the guiding principles for expert system builders is that "in the knowledge lies the power." (See [Feigenbaum, 1977], although the quotation is not literally there.) The rules of an expert system are, of course, the "knowledge" there. Successful (and not so successful) expert systems were designed in an incremental, iterative way. One of the most easily identifiable phases in this process, sometimes called tuning, consists of the refinement of the strengths associated to the rules. This phase is not recorded and discussed as well as it deserves in the literature, perhaps because of the natural inclination to present expert systems as polished products without much concern for the process by which they were obtained, especially when the process is informal and heuristic.

The tuning of Prospector is briefly documented in [Gashnig, 1982]. In a section entitled "Use of Performance Evaluation in Refining a Model," Gashnig describes a "calibration exercise," in which a series of cases given by an expert are run on Prospector. The expert is asked to provide a strength for each intermediate hypothesis as well as for the conclusion. The differences between the strengths given by the expert and those computed by Prospector are tabulated. Since the differences over several cases are greatest for only a few hypotheses, Gashnig (a knowledge engineer for the Prospector expert system) and the expert concentrate on the rules concluding those hypotheses.

In the development of the Graduate Course Adviser, the rules were tuned by a committee during a seminar. Moreover, test cases were run and data was collected with the intention of using the results to modify the strengths associated with the rules. The students who took the tests were asked for permission to check which courses they actually enrolled in, in order to compare the schedule chosen by them and their adviser to the one suggested by the system. An attempt was made to modify certain numerical parameters in order to improve the performance of the expert system on a collection of known cases. This seems to be a typical way to proceed in the construction of an expert system. (See also [Hayes-Roth et al., 1983].)

The implicit assumption underlying the process of tuning the rules is that the rules derived from the expert are correct except for a (usually small) variation in the strength parameters.

We cannot expect the team of expert and knowledge engineer to assign correctly the numerical parameters associated to the rules. Instead, I propose that these parameters be refined by experience on concrete test cases of known solution. This overcomes the bind noted at the end of the previous section.

Knowledge refinement may be considered a middle-ground approach between traditional knowledge engineering techniques and automated "learning from examples" approaches [Michalski and Chilauski, 1980a; 1980b]. One can envision a spectrum of approaches to knowledge acquisition that range from statistical techniques used to derive knowledge directly from data, to total reliance on the expert. An advocate of this second approach was the late John Gashnig, who used to claim that an expert system should duplicate as closely as possible the performance of the human expert interviewed by the knowledge engineer.

Peter Cheeseman argues that one should completely ignore experts. "Currently, the main bottleneck in building expert systems is the time necessary for the expert and the knowledge engineer to find and debug a useful set of rules for a given domain. In some domains there are no experts, or there is little agreement among the experts. Even when there is a suitable expert for a domain, there is considerable evidence that such experts estimate subjective probabilities (or other uncertainty measures) with poor accuracy or even produce inconsistent values. Given these problems [I propose] to bypass the expert and induce the required information directly from the data" [Cheeseman, 1984, p.115].

I would argue that ignoring an expert because it is hard to extract information from him is akin to throwing away the baby with the bath water. Different workers in the expert system field have different views of what "knowledge" means [Brodie, 1984]. Still, one pragmatic way to distinguish knowledge from data is that data can be supplied and maintained by "clerks," while knowledge is supplied by "experts"!

This is not to be taken lightly, because there is evidence that experts structure information differently than novices. The authors of a study entitled "Expertise in Problem Solving" believe that their experiments show that novices and experts employ similar search strategies, but using different knowledge bases [Chi et al., 1981]. (In fact, the novices sometimes seem to use more sophisticated search strategies.) The study reports findings on the way in which novices (two students who have just completed a course in physics) and experts (two physics professors) solve several physics problems from a textbook. "A significant focus for understanding expertise is investigation of the characteristics and influence of organized, hierarchical knowledge structures that are acquired over years of learning and experience" [Chi et al., 1981, p.8]. For example, the novices' "knowledge is organized around dominant objects (such as an inclined plane) and physics concepts (such as friction) mentioned explicitly in the problem statement. Experts, on the other hand, organize their knowledge around fundamental principles of physics (such as Conservation of Energy) that derive from tacit knowledge not apparent in the problem statement. An individual's 'understanding' of a problem is dictated by knowledge of such principles. Hence, during 'qualitative analysis' of a problem, an 'expert' would 'understand' a problem much better than a novice, because he 'sees' the underlying principle" [Chi et al., 1981, p.85].

Expert-derived rules can be considered to form an organized knowledge structure. Usually, they are layered in a hierarchy corresponding to a hierarchy of concepts.

It is also possible to argue the usefulness of extracting rules from experts by noting that machine learning techniques, despite early illusions [Nilsson, 1965] can be applied successfully only to domains exhibiting a high degree of structure. This is due in part to pragmatic reasons, in part to theoretical ones [Gold, 1967]. "Significant learning at a significant rate presupposes some significant prior structure" [Minski and Papert, 1969, p.16]. Using the structure of rules and their interconnection makes the knowledge refinement task feasible in principle.

Much has been written on the way certainty factors are to be combined. A summary (although not totally up to date) is in [Prade, 1983]. The work of Håjek is also to be mentioned, because it is an axiomatic treatment of the subject: he defines a number of axioms that should be satisfied by any calculus of certainty factors and derives certain consequences from them [Håjek, 1982]. The works by Politakis and Rada are the ones that most resemble that described in this thesis.

Politakis has built a program, SEEK, that helps the expert refine the rules by applying heuristics that assign credit and blame to individual rules on the basis of their performance on a collection of cases. Politakis' work is of an applied nature [Politakis, 1982; Politakis and Weiss, 1980; 1984]. He is more concerned with building a system that refines a rule base for a very simple architecture than with questions of method or complexity. Only one layer of rules (the ones that conclude about goals) is allowed to have certainty factors, and their value must be chosen among three (possible, probable, and definite). The first limitation implies that no calculus of combining evidence is necessary. It may therefore be surprising that even such a simple rule base can contain errors after the individual rules are extracted from experts. Still, several test cases with sample rule bases were diagnosed wrongly and SEEK proved useful in correcting the rules. SEEK was used on rule bases containing from 150 to 1000 rules.

Rada's motivation for his work on weight refinement is similar to mine. Rada developed an expert system for the interpretation of computerized axial tomography of the brain. "The rules that were first developed for CT interpretation did not meaningfully interact. Of the several problems, one of the simpler ones stemmed from propagation of weights. What had seemed a reasonable weight for a rule when viewed in isolation, proved otherwise when the rules interacted with others" [Rada, 1984]. Rada's work started and continued independently of mine. Rada recently became interested in refining the strength of links between concepts in a text retrieval system [Rada et al., 1985; Forsyth and Rada, 1986]. Rada's algorithms for weight refinement will be discussed later.

Summary of results

In order to summarize the results, it is unfortunately necessary to introduce some preliminary definitions. They will be repeated and expanded upon in chapter 2 ("Modeling").

The thesis deals with the determination and the adjustment of rule weights. The knowledge-based systems we consider operate by chaining rules together. For most of our purposes, a rule has the form IF (P_1&P_2&...&P_n) THEN C WITH ATTENUATION a, where P_1, P_2,...,P_n, and C are weighted propositions and a (called an attenuator) is a number between 0 and 1 inclusive. By weighted proposition, we mean a statement, possibly true or false, with a certainty factor (CF). CFs have values between 0 and 1, inclusive. A combinator is a function from a vector of CF's to a CF that assigns a single number to the conjunction P_1 & P_2 & ... & P_n of premises P_i of the rule. The most used function here is the MIN function. We multiply the combinator output by the attenuation value a to determine the CF associated with the conclusion C of the rule. Many rules may have the same conclusion; their collective input is merged by an integrator function that defines the CF value associated with weighted proposition C in the rule system. One function that is often used as an integrator is MAX. We have also considered probabilistic addition. Moreover, some of the results obtained consider the role played by predicate functions. A predicate function takes the CF of a weighted proposition in the premise of a rule and returns a CF for that same proposition.

We found it convenient to represent knowledge-based systems as graphs, called inference nets, composed of basic building blocks that represent a combinator, an attenuator, or an integrator. For example, the knowledge-based system composed of the two rules IF P_1 & P_2 THEN P_5 WITH ATTENUATION A_1 and IF P_3 & P_4 THEN P_5 WITH ATTENUATION A_2 would be represented as shown in Figure 1.1

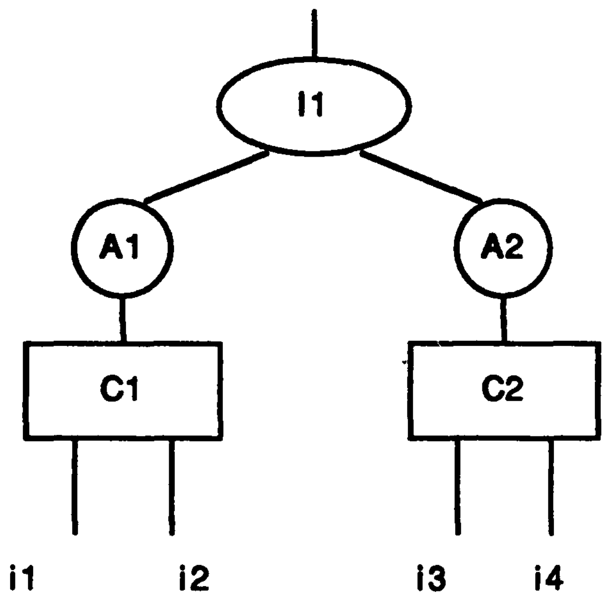


Figure 1.1 An inference net representing two rules

The certainty factors of weighted propositions P_1, P_2, P_3, and P_4 enter input lines i_1, i_2, i_3, and i_4. These values are processed by combinators C_1 and C_2, attenuators A_1 and A_2, and integrator I_1. The output of integrator I_1 is the certainty factor of weighted proposition P_5.

Inference nets realize functions from vectors of CFs to vectors of CFs; a point in the graph of the function is called a test. In the example above, the input vectors have cardinality 4, while the output vector consists of a single CF; a test for the net in the example is a pair composed of vector of input CFs and an output CF.

We assume that the inference net to be built is completely specified, except for the values of the attenuators. The designer of the knowledge-based system must determine (synthesize) or adjust (refine, if estimates of the values are given) these values. We consider two ways in which the designer can learn the values. We call the first model of learning the complete case and the second model the incomplete case. In the complete case, all tests are available; in the incomplete case, only a limited collection of tests is available.

In the complete case, we postulate the existence of a perfect expert, i.e., an oracle that outputs the output part of a test. For example, the perfect expert would be able to diagnose correctly whatever description of a patient were to be presented to it. The patient description is an assignment of certainty factors to a predetermined set of patient descriptors (weighted propositions), while the diagnosis is an assignment of CFs associated with a predetermined set of possible diseases (also weighted propositions). (For inference nets that are trees, this set would include only one disease.)

The results for the complete case can be summarized as follows:

(C1) It appears easy to synthesize attenuations for _trees_ with MIN combinators, MAX integrators, and multiplicative attenuations. In particular, it is easy for real attenuations and real certainty factors, within the usual rounding errors introduced by computer multiplication.

(C2) It is NP-Hard to synthesize internal attenuations for _acyclic graphs_ using MIN, MAX, and multiplicative attenuations.

(C3) It is easy to synthesize _input_ attenuations for the same model as in (C2), in a restricted case.

(C4) It is NP-Hard to synthesize attenuations for _chains_ using MIN, MAX, and some choices of attenuations that are not closed under composition.

(C5) Results similar to C1-C4 hold for probabilistic addition in place of MAX. In particular, it is easy to synthesize attenuations for trees with real multiplicative attenuations.

The results for the incomplete case can be summarized as follows:

(I1) Synthesis of attenuations is NP-Hard even for trees with MIN, MAX, multiplicative attenuations, bounded fan-in to MINs and fixed depth of the tree. Note that this models well some typical rule-based expert systems: bounded number of premises to each rule (independent of the size of the rule base) and short inference chains, also independent of the size of the rule base. In particular, if determining attenuations is hard for this highly constrained model, then finding certainty factors for expert systems using MIN and MAX surely is hard, if done from test cases. We have proved that it is NP-Hard to determine attenuations in this restricted setting.

(I2) The result described in I1 holds with probabilistic addition in place of MAX or in addition to MAX.

(I3) Approximate attenuation synthesis. Let the allowable range of attenuations be 0 to 1, inclusive. It is NP-Hard to determine whether the correct attenuations are closer to 0 or to 1.

(I4) Intermediate hypotheses. It is simple to synthesize attenuations if the tests are augmented to contain the certainty factors of intermediate hypotheses.

(I5) Refinement from almost exact attenuations. Consider a family of trees as in (I1), with estimates for all attenuations. Assume that the estimates are closer to the correct attenuations than an arbitrary given constant. This refinement problem is NP-Hard for any positive value of this constant, however small. We have also found a condition under which a fast algorithm for refinement exists and characterized the behavior of several heuristic algorithms for the general case.

(I6) Heuristic algorithms. We have tried to exploit the concept of influential path for the MIN/MAX case for multiplicative attenuations. In this case, the output of the net is always equal to the attenuated value of one of the inputs. We have proved that the problem of setting attenuations remains NP-Hard even when an influential input is specified for each test.

A different organization of results is used for the summaries in Figures 1.2, 1.3, and 1.4. The first two figures outline the results for the synthesis of rule strengths; the third one summarizes the results obtained for the refinement of rule strengths.

	complete case	incomplete case
chains	trivial but NP-Complete when attenuations are not closed under composition	trivial but NP-Complete when attenuations are not closed under composition
trees	fast algorithm given	NP-Complete even when approximate values only are desired for the attenuations
acyclic graphs	fast algorithm given for determination of one input attenuation with small CF alphabets; determination of internal attenuations is NP-Complete	NP-Complete even when approximate values only are desired for the attenuations

Figure 1.2 Summary of results on synthesis, MIN/MAX

	complete case	incomplete case
chains	trivial but NP-Complete when attenuations are not closed under composition	trivial but NP-Complete when attenuations are not closed under composition
trees	fast algorithm given	NP-Complete even when approximate values only are desired for the attenuations
acyclic graphs	fast algorithm given for determination of one input attenuation with small CF alphabets; determination of internal attenuations is NP-Complete	NP-Complete even when approximate values only are desired for the attenuations

Figure 1.3 Summary of results on synthesis, MIN/p+

	MIN/MAX	MIN/p+
principal paths given for each test	NP-Complete	NP-Complete
almost correct estimates given	NP-Hard	NP-Hard (conjectured)
almost correct estimates given, constant increment	NP-Hard	NP-Hard (conjectured)
almost correct estimates given, approximation sought	NP-Hard	NP-Hard (conjectured)
switch setting given	fast algorithm given	slow algorithm given; conjectured to be NP-Hard

Figure 1.4 Summary of results on refinement

Organization of the thesis

The reader looking for an executive summary could limit himself to reading chapter 1 ("Introduction") up to this paragraph for a presentation of the problem and the results, then read the first section of chapter 10 ("Conclusion") for the author's interpretation of the results.

Chapter 2 ("Modeling") describes the model of expert system used throughout the thesis. It also contains a definition of the two learning models used. Chapter 3 ("Test Generation") describes a problem that is in many ways simpler than synthesis or refinement and solves three variants of it to show how different model parameters affect the complexity of solutions. In chapter 4 ("The Complete Case"), the problem of synthesizing attenuations from perfect experts for inference trees is solved. Chapter 5 ("The Incomplete Case") deals with synthesizing attenuations from tests in inference trees. Chapter 6 ("Chains") considers the problem of synthesizing attenuations for inference chains, when the attenuations are not closed under composition. Chapter 7 ("Graphs") deals with the synthesis problem for inference graphs, in a very restricted setting. Chapter 8 ("Approximations and Intermediate Values") extends some of the results of chapter 5 to the case in which approximate attenuations are acceptable and discusses three different learning models, where cases are enriched with information on intermediate hypotheses. Chapter 9 ("Refinement") considers the problem of attenuation refinement.

The results listed in the previous section are contained in the chapters (and sections, where applicable) of this thesis as follows: (C1) chapter 4, section 3; (C2) chapter 7; (C3) chapter 7; (C4) chapter 6; (C5) chapter 4, section 5; (I1) chapter 5, section 2; (I2) chapter 5, section 3; (I3) chapter 8, section 2; (I4) chapter 8, section 3; (I5) chapter 9, section 3; (I6) chapter 9, sections 2, 4, and 5.

CHAPTER TWO

MODELING

Introduction

The main purpose of this chapter is to define the model of expert system and the learning models that will be used throughout this thesis.

There does not seem to be a single accepted definition of what an expert system is. One extreme view is that "an expert system is simply an application program for which there are many tool kits" [Stonebraker, 1984]. Others would argue that an expert system is just a program that simulates a human expert. Therefore, they list the characteristics, such as performance and ability to explain, that are necessary to expert behavior. Others would consider the programs that are called "expert systems" by most researchers and extract their common features, in order to characterize them. For example, Hayes-Roth [1984, p.264] lists the following "characteristics common to expert systems: they solve very difficult problems as well as or better than human experts; they reason heuristically, using what experts consider effective rules of thumb; they interact with humans in appropriate ways, including the use of natural language; they manipulate and reason about symbolic descriptions; they function with erroneous data and uncertain judgemental rules; they contemplate multiple competing hypotheses simultaneously; they explain why they are asking a question; they justify their conclusions." However, these are behavioral characterizations, whereas this chapter is concerned with modeling from a structural point of view; at this level, precision is possible.

The second section introduces the model of expert system and a graphical notation, and it contains a discussion of the coverage afforded by the model. The third section defines the synthesis and refinement problems and the two learning models that will be studied in the thesis.

A model of expert systems

Rule-based systems are the only ones considered here. (See [Nau, 1983] for a classifications of expert systems that includes other organizations, such as frame-based ones. Strong rule-based components are present also in the most sophisticated commercially available expert system shells.)

The expert systems that we consider operate by chaining rules together. The specific order of rule chaining can be shown to be irrelevant, for most of the results we obtain, but backward chaining is always considered in the examples. A rule has the form IF (P_1 & P_2 & ... & P_n) THEN C WITH ATTENUATION a, where P_1, P_2, ..., P_n, and C are weighted propositions and a is a function from reals to reals. A weighted proposition is a statement, possibly true or false, with a numeric weight, called a certainty factor (CF), that reports the degree of belief that the associated proposition is true. A combinator is a function from vectors of CFs to a CF that assigns a single number to the conjunction P_1 & P_2 & ... & P_n of premises of the rule. An attenuator is a function that maps the CF of the premises of the rule (as given by applying the combinator) to the CF of the conclusion of the rule. Many rules may have the same conclusion; their collective input is merged by an integrator function that defines the CF value associated with weighted proposition C in the rule system. To complete the definition of the rule form described above, we introduce predicate functions. A predicate function takes the CF of a weighted proposition in the premise of the rule and returns a CF for the same proposition.

It is convenient to represent knowledge-based systems as graphs, called inference nets, composed of blocks representing their components. A typical basic subgraphs representing a rule with three weighted propositions is shown in Figure 2.1, where the oval, circle, square, and lozenge represent an integrator, an attenuator, a combinator, and a predicate function, respectively.

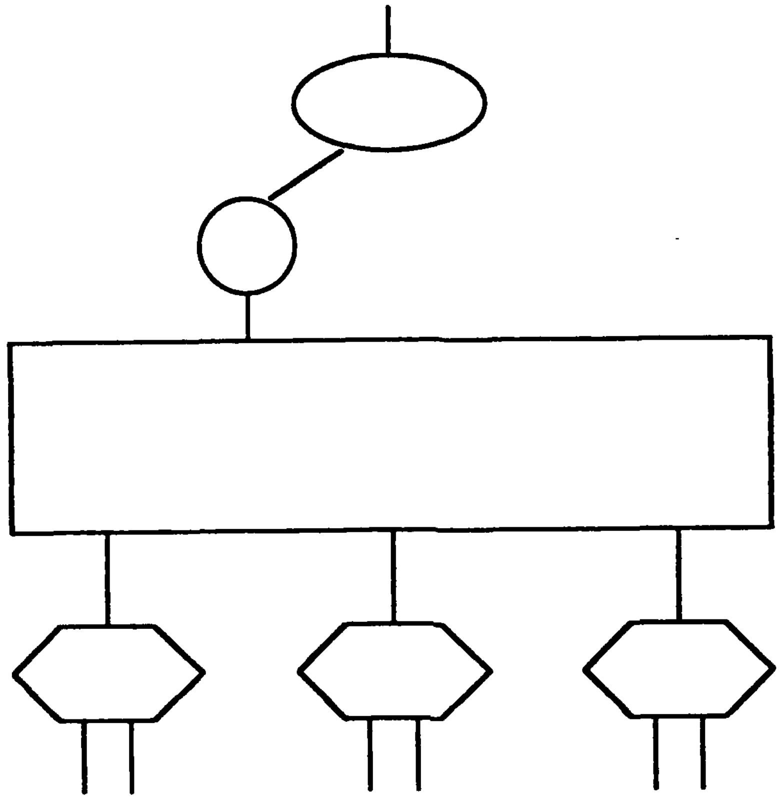


Figure 2.1 Graphical representation of a rule

<u>Example 2.1</u>

The knowledge-based system composed of the two rules IF P_1 & P_2 THEN P_5 WITH ATTENUATION A_1 and IF P_3 & P_4 THEN P_5 WITH ATTENUATION A_2 would be represented as shown in Figure 2.2.

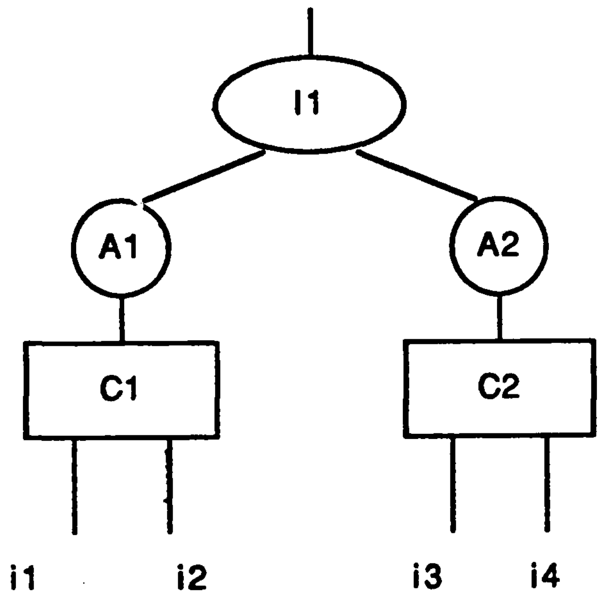


Figure 2.2 An inference net representing two rules

In this example, there are no predicate functions. The certainty factors of weighted propositions P_1, P_2, P_3, and P_4 flow from the input lines i_1, i_2, i_3, and i_4 through combinators C_1 and C_2, attenuators A_1 and A_2, and integrator I_1. The output of integrator I_1 is the certainty factor of weighted proposition P_5.

(End of example)

<u>Example 2.2</u>

In this example, we represent a small portion of the rule base of the Graduate Course Adviser [Valtorta et al., 1984]. The rules (translated into English) that are represented are:

IF the student expresses interest in the analysis of algorithms
THEN the student in interested in theory WITH ATTENUATION .7

IF the student had math as a major
THEN the student is interested in theory
WITH ATTENUATION .8

IF the student is interested in theory
AND the student expresses interest in mathematics
THEN the student should take MTH251
WITH ATTENUATION .9

The graphical representation is shown in Figure 2.3.

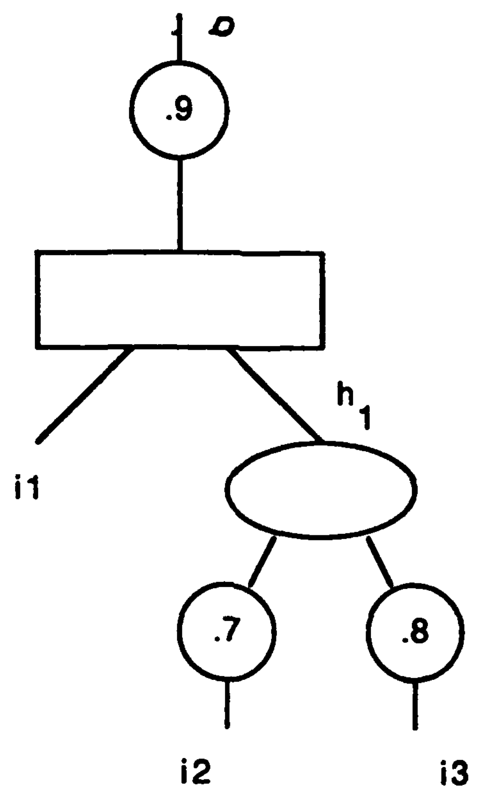


Figure 2.3 A small portion of the rule base of the Graduate Course Adviser

In this example, the CF flowing on i_1 is the CF for the proposition "the student expresses interest in mathematics." The CF flowing on i_2 is the CF for the proposition "the student expresses interest in the analysis of algorithms." The CF flowing on i_3 is the CF for the proposition "the student had math as a major." The CF flowing on h_1 is the CF for the proposition "the student is interested in theory." The CF flowing on o is the CF for the proposition "the student should take MTH251." Note that predicate functions and integrators that are the identity function are not indicated in the picture.

(End of Example)

For the same rule base, one can have different nets depending on the facts and the information provided by the user only if variables appear in the rules.

The graphical representation is analogous to that described by Slagle [1984]. Also, "model diagrams" are the preferred way to represent Prospector's rule bases [Reboh, 1981]. The major commercially available expert system shells, such as KEE [Richer, 1986] heavily use graphics to represent their knowledge base. However, the graphs are used to display interconnections of objects (e.g., frames, units) rather than rules.

There are rule-based expert systems that cannot be presented graphically as indicated. An example is Ponderosa. "Ponderosa represents a departure from current plausible inference systems because, although it still deals with uncertain assertions and relations, _it does not attempt to propagate validity measures of any kind_ [italics mine]. Instead, it follows the approach of trying to identify internally consistent subsets of the information given to it. The merit of any such division is then established as a function of the validities of the assertions that were not included" [Quinlan, 1983b, p.141]. In fact, the graphical representation is suitable only to represent _truth-functional_ expert systems, as defined by Ruspini [1982, p.88]. A rule-based expert system is truth-functional if the certainty factor of an hypothesis depends only on the certainty factors of premises in rules that conclude the hypothesis. Ponderosa is not a truth-functional expert system, because the certainty factor associated with a hypothesis depends on the strengths of the rules that were not used to conclude the hypothesis.

Different choices for combinators, attenuators, integrators, and predicate functions allow one to model most existing rule-based expert systems.

Example 2.3

With the choice of min and max for combinators and max for integrators, one can model the systems that use fuzzy-set theory formulae, such as AL/X [Quinlan, 1983a, p.258], and EXPERT [Weiss and Kulikowski, 1979].

(End of example)

Example 2.4

The choice of min and max for combinators and probabilistic addition for integrators lets one model MYCIN-like systems, like MYCIN itself and the GCA. The probabilistic addition (or probabilistic sum) of a and b (both being between 0 and 1) is a+b-ab and is indicated a[p+]b.

(End of example)

In almost all expert systems, the certainty factor associated to the conclusion of a rule is given by the CF of the premise multiplied by a coefficient between 0 and 1. These coefficients will be called weights. In the ideal case of real weights and real CFs between 0 and 1, these weights have the property of being closed under composition: the same attenuation that can be obtained by applying a sequence of attenuations can also be obtained by applying a single one of value equal to the composition of the ones in the sequence.

Synthesis, refinement, and learning models

We view the inference net as the realization of a mapping between a vector of certainty factors (on the input lines of the net) and a vector of certainty factors (on the output lines of the net). The function to be realized is unknown to the designer of the expert system. The knowledge acquisition process is the process of discovering this function. The knowledge engineer has several ways of discovering it, such as interviews with experts (i.e., use of knowledge elicitation techniques), generalization from cases of the problem to be solved, and analysis of protocols of expert consultations.

In this dissertation, we assume that only the attenuation values have to be discovered; the net is given. We consider mainly two problems, the synthesis problem and the refinement problem. In the synthesis problem, no estimates are given of the values of the attenuations. In the refinement problem, estimates of the attenuations are given and the designer must adjust the given values. Both can be considered to beinstances of a learning problem.

Two main learning models are considered. Before describing them, we introduce a definition.

Definition 2.1 A test is a pair (input part, output part), where input part is a vector of CFs corresponding to the inputs of the inference net and output part is a vector corresponding to the outputs of the inference net.

Only inference nets with a single output will be considered almost exclusively. For simplicity, in this case output part is considered a single value rather than a singleton vector.

The first model of learning is called the _complete case_ and the second model the _incomplete case_. In the complete case, all tests are available; in the incomplete case, only a limited collection of tests is available.

In the complete case, we postulate the existence of a _perfect expert_, i.e., an oracle that outputs the output part of a test, when given the input part of a test. For example, the perfect expert would be able to diagnose correctly whatever description of a patient were to be presented to it. The patient description is an assignment of certainty factors to a predetermined set of patient descriptors (weighted propositions), while the diagnosis is an assignment of certainty factors associated with a predetermined set of possible diseases (also weighted propositions). For inference nets that are trees, this set would include only one disease.

CHAPTER THREE

TEST GENERATION

Introduction

After defining and discussing the model of expert systems in the previous chapter, we consider the synthesis and refinement problem for various expert systems, i.e., for various choices of the model parameters. But a problem that is in various ways "simpler" than knowledge refinement will be introduced first. The problems that follow are motivated by a desire to start from simpler, although not necessarily useful, situations, in order to gain understanding.

The test generation problem

The generic test generation problem consists of finding tests that are handled correctly by a given, completely specified, rule base.

Several specific cases of the test generation problem will now be considered. The results will illustrate how different selections of net topology, allowable attenuators, and allowable predicate functions affect the complexity of problems involving functional rule-based expert systems.

Decision problems will be described following the format defined by Garey and Johnson [1979, p.18], which consists of three parts: a name, a generic instance of the problem in terms of various primitive components, such as graphs and numbers, and a yes-no question asked in terms of the generic instance. The generic instance will be referred to as the instance, for short.

Problem name. Test generation, acyclic graph with two predicate functions (TGN).

Instance. Network topology is an acyclic graph with a single goal. Integrators are two-input max functions. Combinators are two-input min or two-input max functions. Certainty factors are 0 and 1. (I.e., the CF alphabet is {0,1}.) Attenuations are identity (mapping 0 into 0 and 1 into

1). Predicate functions are identity (mapping 0 into 0 and 1 into 1) and complementation (mapping 0 into 1 and 1 into 0). (This models, for example, the predicate function thoughtnot used in MYCIN.) Therefore, the attenuators are not monotonically non-decreasing, but they are closed under composition. The instance is a net satisfying the criteria described above.

Question. Is there an input test vector such that the output of the net is 1?

Theorem 3.1 TGN is NP-Complete

Proof (1) The problem is in NP, because it takes only polynomial time to run a test through a tree, i.e., to compute the output CF from the input CF's, and the number of tests is exponential in the number of input nodes and therefore no more than exponential in the input size. (2) Boolean satisfiability can be reduced to TGN. Consider any formula that uses OR, AND, and negation as logical connectives. Represent the resulting formula as an acyclic network. This is a relatively standard way of representing formulas, and the conversion from list format to network format can be done simply in polynomial time. For example, -[(A&B)vC] (where "-" stands for "not") is represented as the network in Figure 3.1. The network has the properties described above, when OR's are interpreted as integrators or combinators, AND's as combinators, and NOT's as predicate functions. Each input to the net corresponds to a variable in the formula. An input to the net produces an output of 1 if and only if the original formula is satisfiable.

(End of proof)

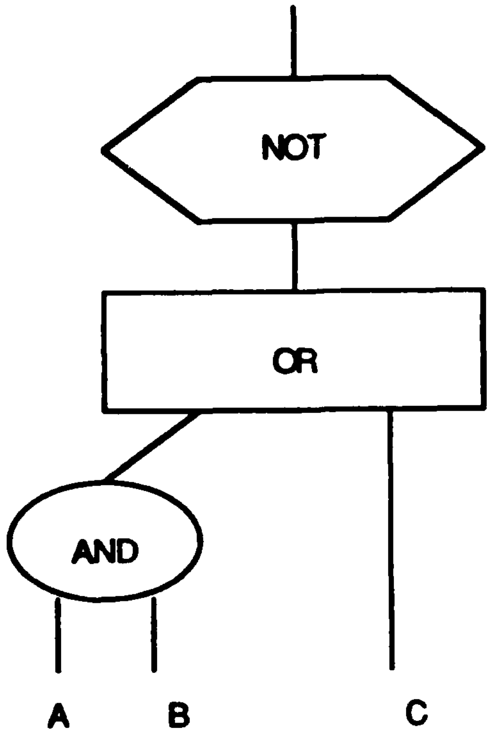


Figure 3.1 Network representing ¬[(A&B)vC]

Note that the result holds even if the instance is generalized in one or both of the following two ways: (1) more than two inputs are allowed for combinators and integrators; (2) only MIN is allowed as an integrator.

Problem name. Test generation, tree (TGT).

Instance. The network is a tree. The choices for all other parameters are the same as in problem TGN. The instance is a net satisfying the above criteria.

Question. Is there an input part such that the output of the net is 1?

<u>Theorem 3.2</u> TGT can be solved in constant time.

<u>Proof</u> The answer to the question in TGT is always "yes." Consider each path from the root to a leaf independently and

set the input value of the input to the leaf to 0 if there is an odd number of attenuations having value "complement" on the path; set the input to 1 otherwise.

(End of proof)

Problem name. Test generation, acyclic graph, no predicate functions (TGM).

Problem instance. In TGM, no predicate functions are available. (In other words, the only available predicate function is identity.) All other parameters are the same as in TGN.

Question. Is there an input part such that the output of the net is 1?

<u>Theorem 3.3</u> TGM can be solved in polynomial time.

<u>Proof</u> The following polynomial-time algorithm solves TGM. Propagate the input vector of all 1's through the net. If the output is 1, the answer is "yes," otherwise it is "no."

(End of proof)

TGM can be generalized to different CF alphabets, combinators, and integrators. Theorem 3.3 keeps holding as long as the only predicate function is identity and the attenuators are monotonically non-increasing.

This chapter has shown the effect of net topology and allowable attenuators and predicate functions on the complexity of test generation.

CHAPTER FOUR

THE COMPLETE CASE

Introduction

In the complete case, we model an expert as a perfect expert, as defined in Section 6 of Chapter 2. The name, complete case, stems from the fact that the input/output function to be realized by the rule-based system is completely available, rather than being available only through samples.

All results in this chapter concern trees. The first section shows that only a fraction of the attenuations in a tree with MIN/MAX attenuators and combinators and attenuators that are closed under composition can be set independently. An algorithm to synthesize the attenuators in the case in which they are multiplicative is then given. The second section presents an algorithm to redistribute the attenuators so that they are as close as possible, according to a suitable metric, to given ones--typically, expert-given ones. The third section contains an algorithm to synthesize attenuators in trees with MIN combinators and probabilistic sum integrators.

The more general (and more difficult) problem of computing attenuations for graphs is considered, in a very restricted case, in Chapter 7.

Independent attenuations

This is a case in which only a fraction of the attenuations are independent of each other. It is useful to know how many attenuations can be set independently, in order to simplify knowledge refinement and to evaluate the power of a network, i.e., the number of different input vectors that produce different outputs.

Definition 4.1 A binary tree is said to be complete if for some integer k, every vertex of depth less than k has both a left child and a right child and every vertex of depth k is a leaf. A complete binary tree of height k has exactly $2^{k+1}-1$ vertices [Aho et al., 1974].

Theorem 4.1 Let an inference tree of depth d be given. (See Figure 4.1.) Let $g_1, \ldots, g_{kp}$ be choice functions, like min and max. Let $f_1, \ldots, f_{kp}$ be the corresponding attenuations. For each i/o pair, $((x^i_1, \ldots, x^i_{2p}), o^i)$, where i is the i/o pair index, at least one of the following p constraints holds: $f_1(f_{j2}(f_{j3}\ldots(f_{jd}(i_{jd}))\ldots)=o^i$, where $2j_k=j_{k+1}$ or $2j_k+1=j_{k+1}$ and $j_1=1$.

Proof

The output of the tree is the output of f_1.

The proof is by induction on the depth of the tree.

Basis The output of leaf attenuations is $f_j(i_j)$, where j is a leaf index.

Inductive step Assume that the output of all attenuations at depth d-n, where d is the depth of the whole tree (here, the fact that the tree is complete is used), is of the form $f_{j1}(f_{j2}\ldots(f_{jn}(i_{j1})\ldots)$, where $2j_k=j_{k+1}$ or $2j_k+1=j_{k+1}$. Then the output of attenuations at depth d-n-1 ("one level up the tree") is $f_{j0}(f_{j1}\ldots(f_{jn}(i_{jn})\ldots)$, where $2j_0=j_1$ or $2j_0+1=j_1$. This is true because g_{j0}, the choice function corresponding to f_{j0} outputs either $f_{2j0}(\ldots(f_{jn}(i_{jn})\ldots)$ or $f_{2j0+1}(\ldots(f_{jn}(i_{jn})\ldots)$.

(End of proof)

The proof carries over, with obvious modifications, to incomplete trees, but it has been described only for complete trees, for simplicity.

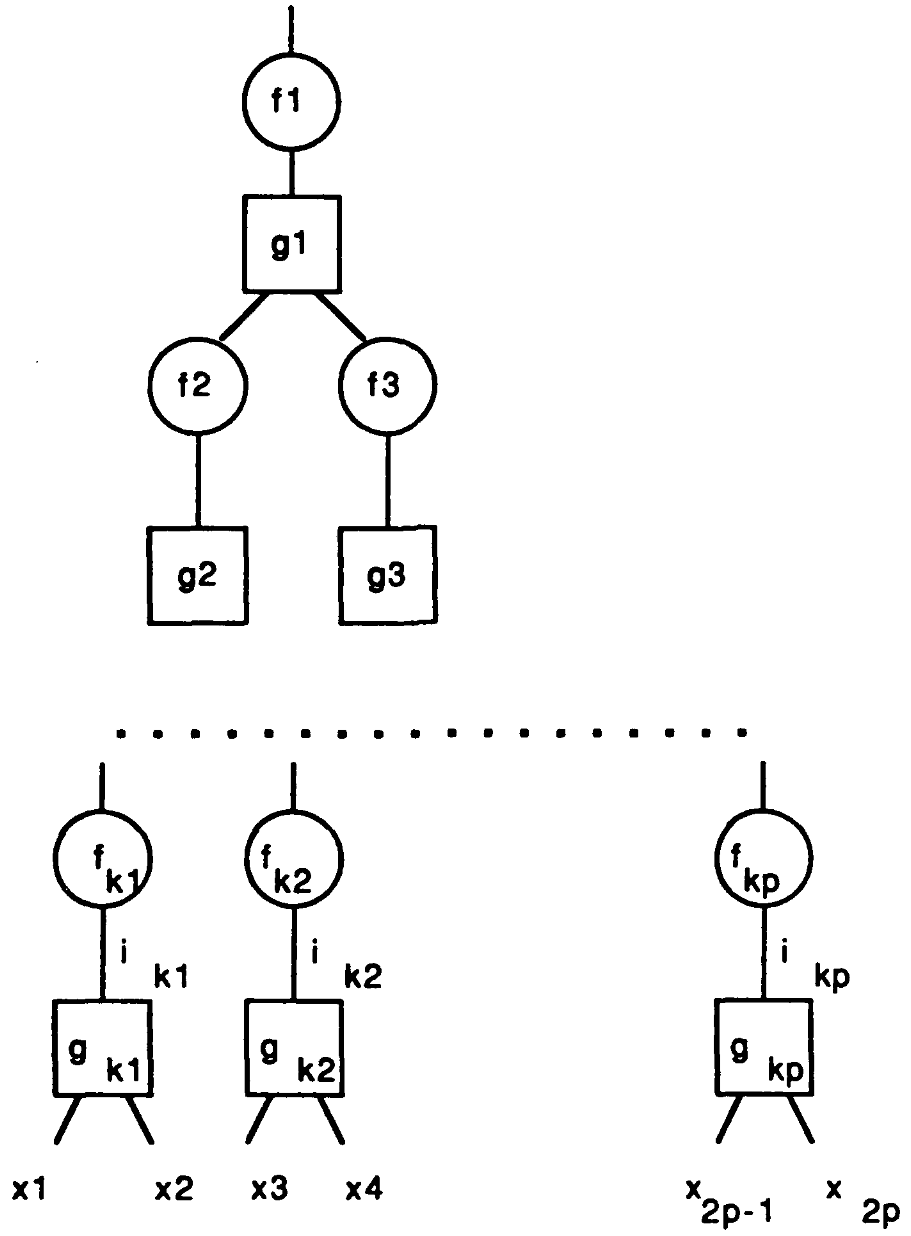


Figure 4.1 Complete inference tree of depth d, with p inputs

<u>Definition 4.2</u> An <u>input attenuation</u> is an attenuation closest to an input of an inference net. All attenuations that are not input attenuations are <u>internal attenuations</u>. The attenuations that are closest to the outputs of the net are <u>output attenuations</u>. In the case of trees, input attenuations

are also called <u>leaf attenuations</u>.

If the attenuations are closed under composition, this result implies that in a MIN/MAX tree, any i/o function that can be obtained by setting all attenuation independently of each other can also be obtained by setting the internal attenuations to identity and setting the input attenuations independently.

This can be stated as follows: all tests that can be realized by any assignment of attenuations can also be realized by an assignment of attenuations with the following properties:
(1) the leaf attenuations are the composition of the attenuations at the (unique) path from the leaf to the root;
(2) all other attenuations are the identity function.

This result can be extended to the case in which the combinator or integrator function in the tree is a linear function. The case in which the linear function is addition throughout the tree will be illustrated first. The same numbering scheme as in Figure 4.1 will be used.

<u>Theorem 4.2</u>

Let an inference tree of depth d be given. (See Figure 4.1.) Let $g_1, \ldots, g_{kp}$ be the function sum. Let $f_1, \ldots, f_{kp}$ be the attenuations corresponding to the sums. For each i/o pair, $((x^i{}_1, \ldots, x^i{}_{kp}), o^i)$, where i is the i/o pair index, the following constraint holds:
$f_1(f_{j2}(f_{j3} \ldots (f_{jd}(i^i{}_{jd})) \ldots) = o^i$, where the sum is over all possible distinct choices of $j_2, j_3, \ldots, j_d$ satisfying $j_1=1$, $2j_k=j_{k+1}$ or $2_j{}^k{}_{+1=j}{}^{k+1}$.

<u>Proof</u>

The output of the tree is the output of f_1. The proof is by induction on the depth of the tree.

<u>Basis</u> The output of leaf attenuations is $f_j(i_j)$, where j is a leaf index.

<u>Inductive step</u> Assume that the output for test i of subtrees whose root is at depth d-n, where d is the depth of the whole tree, is of the form

$$\sum (f_{j1}(f_{j2} \ldots (f_{jn}(i^i{}_{jn}) \ldots),$$

where the sum is over all possible choices of $j_1, j_2, \ldots, j_n$ satisfying $2j_k=j_{k+1}$ or $2j_k+1=j_{k+1}$. Then the output of the subtrees whose root is at depth d-n-1 ("one level up the tree") is $\sum (f_{j0}(f_{j1} \ldots (f_{jn}(i^i{}_{jn}) \ldots)$, where the sum is over all possible choices of $j_0, j_1, j_2, \ldots, j_d$ satisfying $2j_k=j_{k+1}$ or

$2j_k+1=j_{k+1}$. This is true because of the distributive property of f_i's over sum: $(f_i+f_j)f_k = f_if_k+f_jf_k$ and of the fact that f_i's are closed under composition: $f_i^jf_k^k = f_j$, for some k for every i,j.

(End of proof.)

Example 4.1

Consider the simple tree in Figure 4.2 and the tests expressed in the table below.

	i_1	i_2	i_3	i_4	o
T_1	3	1	4	2	1
T_2	4	4	3	3	2
T_3	3	3	4	4	2

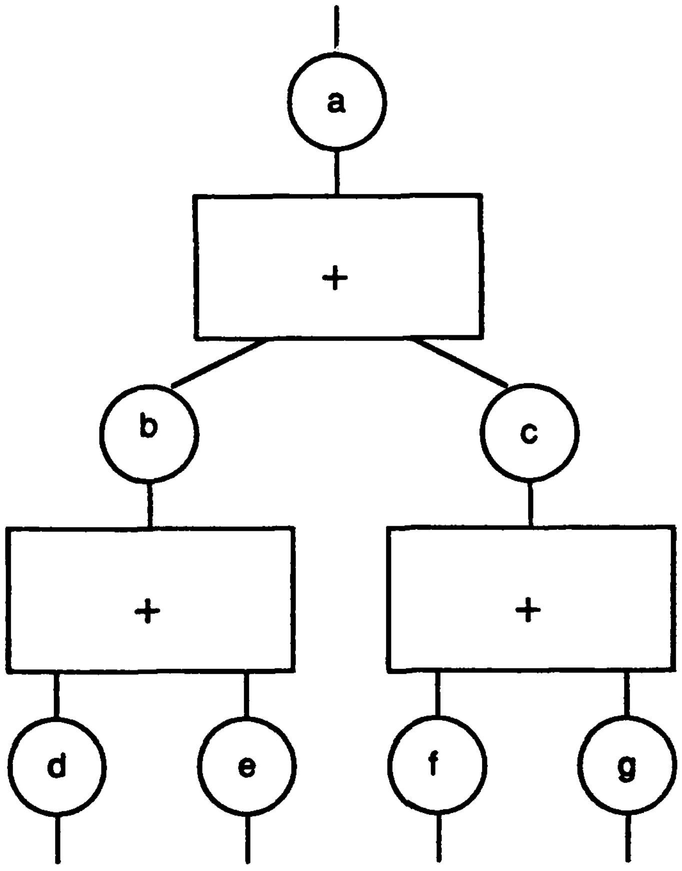


Figure 4.2 An inference tree with sum integrators and combinators

The tests constrain the attenuations in the following way:

(T_1) $[(3d+e)b+(4f+2g)c]a = 1$
(T_2) $[(4d+4e)b+(3f+3g)c]a = 2$
(T_3) $[(3d+3e)b+(4f+4g)c]a = 2$

That is:

(T_1) $3dba+eba+4fca+2gca = 1$
(T_2) $4dba+4eba+3fca+3gca = 2$
(T_3) $3dba+3eba+4fca+4gca = 2$

Since d and e are always multiplied by ba, and f and g are always multiplied by ca, this system has a solution if and only if the following one, where a=b=c=1, does:

(T_1) $3d+e+4f+2g = 1$
(T_2) $4d+4e+3f+3g = 2$
(T_3) $3g+3e+4f+4g = 2$.

(End of example.)

Theorem 4.2 holds with obvious modifications even when linear functions other than sum are used, i.e., it holds when functions g_i are used such that $f_1(g_i(f_2,f_3))=g_i(f_1f_2,f_1f_3)$. Note that the linear functions used do not need to be the same throughout the tree. For example, sum may be used as an integrator and subtraction as a combinator. This last statement will be illustrated with a simple example.

<u>Example 4.2</u>

The tests in this example are the same as in Example 4.1. The network is illustrated in Figure 4.3.

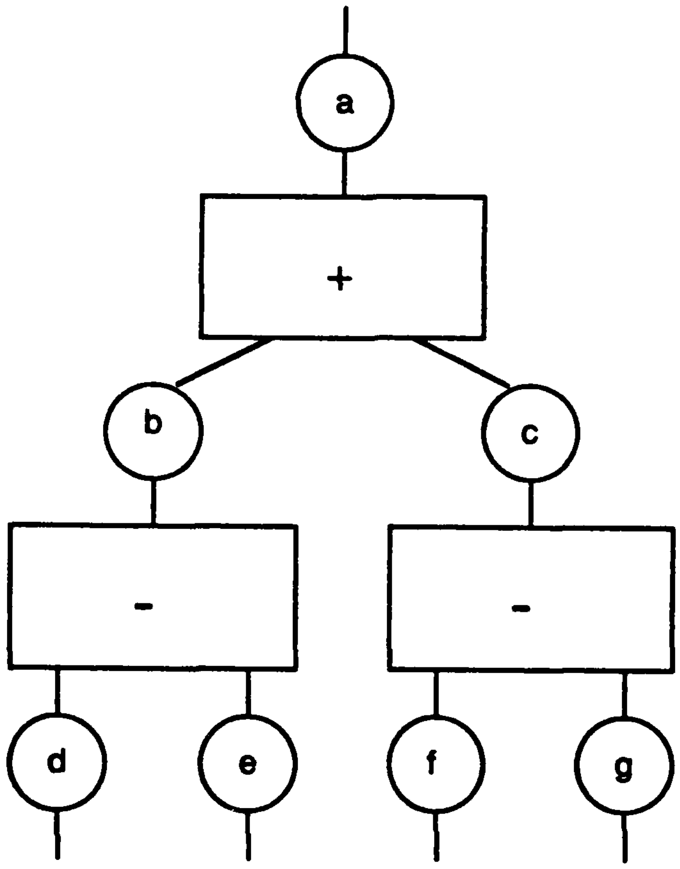


Figure 4.3 Inference tree with sum integrators and subtraction combinators

The tests constrain the attenuations in the following way:

(T_1) $[(3d-e)b+(4f-2g)c]a = 1$
(T_2) $[(4d-4e)b+(3f-3g)c]a = 2$
(T_3) $[(3b+3e)b+(4f+4g)c]a = 2$

That is:

(T_1) $3dba-eba+4fca-2gca = 1$
(T_2) $4dba-4eba+3fca-3gca = 2$
(T_3) $3dba-3eba+4fca-4gca = 2$

As in Example 4.1, since d and e are always multiplied by ba, f and g are always multiplied by ca, this system has a solution if and only if the following one, where a=b=c=1, does:

(T_1) $3d-e+4f-2g = 1$
(T_2) $4d-4e+3f-3g = 2$
(T_3) $3g-3e+4f-4g = 2$

(End of example)

Synthesis of attenuations: MIN/MAX case

Now, consider the task of synthesizing multipliers (a special kind of attenuators) from a complete test set.

Definition 4.3 A path is sensitized if the net output is equal to the input to the leaf of that path attenuated by the attenuations on the path. (Recall that one can consider only the leaf attenuations as having value different from 1.) A sensitized path is called an influential path or a principal path. The input to a principal path is called a principal input or influential input. (Note that in the case of ties there can be more than one principal paths and inputs.)

Whether we can always sensitize a path in the complete case depends on the model we use. To sensitize a path (in a tree, a branch), the values entering a MAX box must be "low enough" to let the value from the sensitized branch go through; the values entering a MIN box must be "high enough" to let the value from the sensitized branch go through. (See Figure 4.4.) 0's are low enough values, but there may not be any value which is high enough, if an attenuation on a non-sensitized branch is 0. We can test for a path that cannot be sensitized by switching the input to the path from highest (1) to lowest (0). If no change occurs, a 0 is coming from another path in input to a MIN box, i.e., the path cannot be sensitized. But it may take an indefinitely long time to find an input which is low enough to sensitize a path when the outcome of the test is that a path can be sensitized, because the attenuation on another branch may be arbitrarily small.

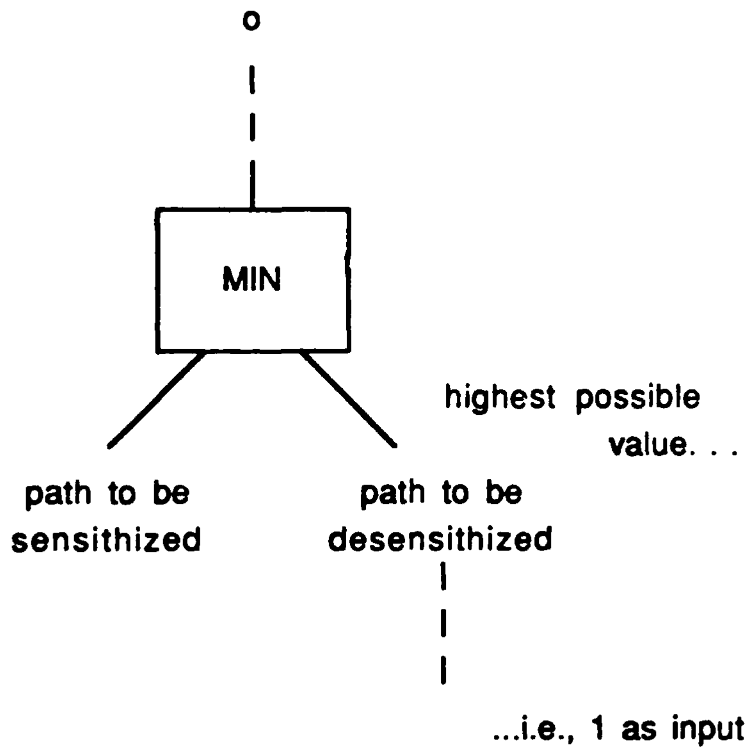


Figure 4.4 Sensitizing a path

This problem disappears if there is a constant min such that all input CF's are greater than it. This seems to be practically an acceptable limitation. More precisely, the range of input values has a min value, but no such limitation is put on the domain and range of the attenuators: the attenuators can produce values that are smaller than the min input value. This choice is motivated by the fact that input values, which are provided by people, or possibly sensors, are unlikely to be arbitrarily low, whereas it may be useful to allow attenuations, which, to some extent, could be synthesized, to have real-valued domain and range.

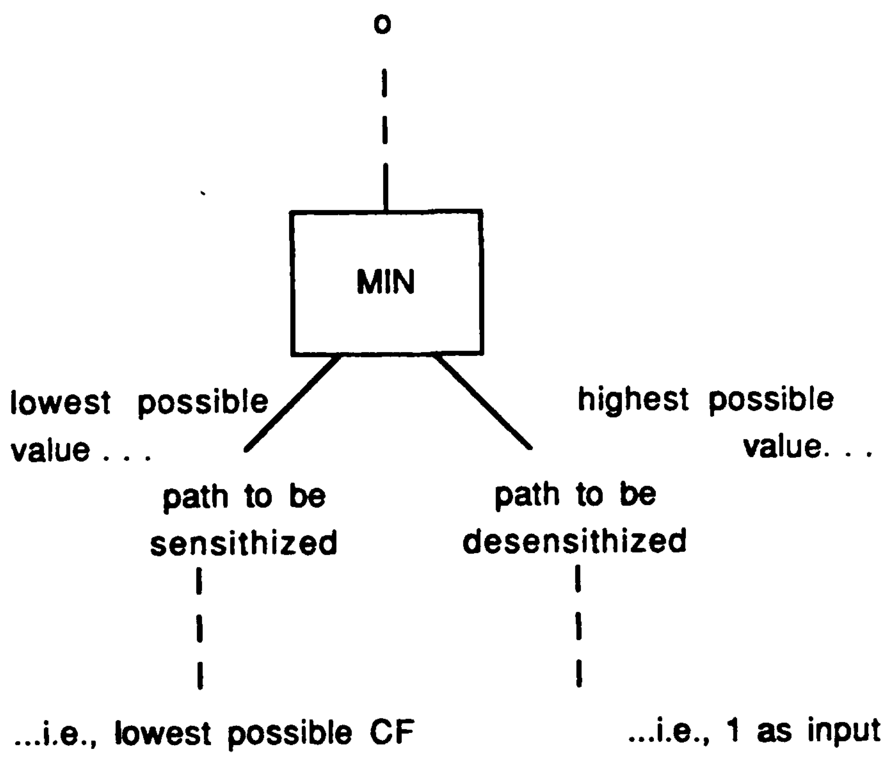

o
MIN
lowest possible value . . .
highest possible value. . .
path to be sensithized
path to be desensithized
...i.e., lowest possible CF
...i.e., 1 as input

Figure 4.5 Sensitizing a path, when there is a lowest input CF

An algorithm to synthesize multipliers in a tree can now be given, for the case in which the condition summarized in Figure 4.5 holds. The algorithm uses exactly one test for each attenuation to be sensitized. The test sensitizes the path on which the attenuation lies. The value of the attenuation is set by dividing the test output by the input to the path.

It is simple to modify the algorithm for the case in which the following non-identity predicate functions are present: 1 is mapped to min and min is mapped to 1, and intermediate values are mapped in such a way that, if a>b, a is mapped into a smaller value than b.

Algorithm to redistribute attenuations

Once weights have been determined, by using the algorithm in the previous section, which synthesizes all weights at the leaves of a tree, they should be redistributed to the internal nodes, to be close to the attenuations given by the expert. This is desirable to please the expert, who is likely to prefer editing rules close to the ones elicited from him, rather than rules that perform correctly on training cases, but which, on an individual basis, are not recognizable. This is consistent with the basic, although hardly formalizable, principle, that rules constitute a form of knowledge organization, which should not be disposed of lightly, lest we lose anything to refine.

Redistribution can be achieved with process truncation error $O(d^2)$, where d is the maximum of the errors in the weights to be refined, by linearizing the optimization problem "minimize the Euclidean distance between the expert-given weights and a set of correct ones," and solving the linearized problem by quadratic programming.

While quadratic programming is NP-Complete [Sahni, 1974], the quadratic programs that we consider here are convex, and therefore they can be solved in polynomial time [Kozlov et al., 1979]. However, it is not suggested that Kozlov, Tarasov and Kachiyan's algorithm be used in practice. The common algorithm [Gottfried and Weisman, 1973, p.214] reduces the quadratic program to a set of linear ones, but only exactly one of them must be solved unless the correct attenuation for some rule is either 0 or 1. This can certainly be considered a degenerate case, and therefore the common algorithm can be used with confidence.

To make the above discussion more precise, consider that the leaf attenuations have been exactly determined. Say that there are p leaves in the tree and that the depth of the tree is m. This means that there are p products of m attenuations set equal to the leaf attenuations: p multiplicative constraints of m variables each. The cost function is the distance between correct attenuations and expert-given ones. Rewrite each attenuation as the sum of the expert-given attenuation plus an increment: $x_i = \underline{x}_i + d_i$. Multiply the attenuations on the same branch and drop the terms that are not constant or linear in the d_i's. We now have a quadratic program: linear constraints and quadratic cost function (the sum of d_i^2). Of course, it must be that $0 <= \underline{x}_i + d_i <= 1$. The following example illustrates this procedure.

Example 4.3

Consider the tree shown in Figure 4.6

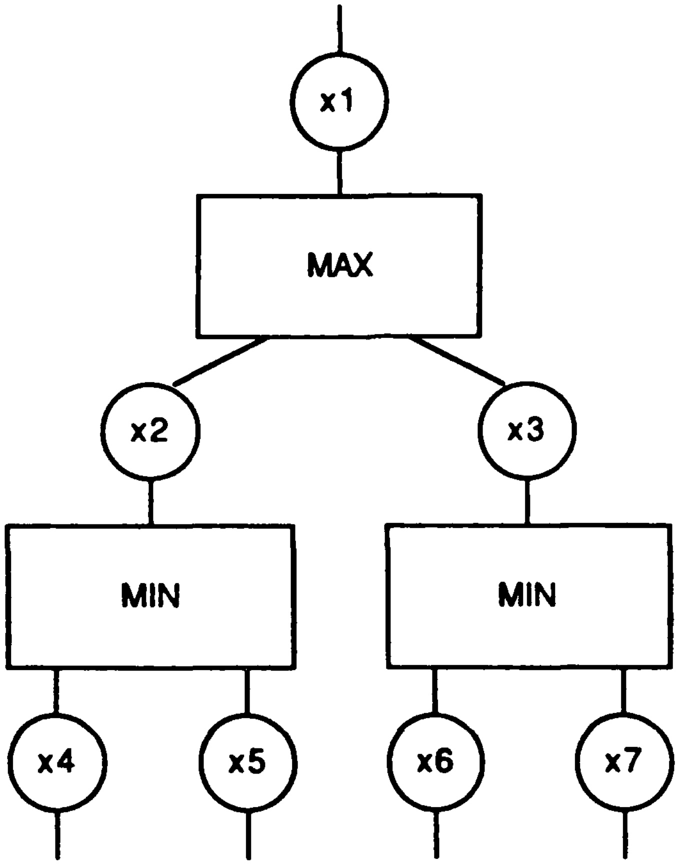


Figure 4.6 An inference tree with redundant attenuators

Assume that it has been determined that the correct values of the leaf attenuations are $x_4=x_4^c$, $x_5=x_5^c$, $x_6=x_6^c$, $x_7=x_7^c$ (and all other attenuations have value 1). The expert-given attenuations have value x_1^e, x_2^e, x_3^e, x_4^e, x_5^e, x_7^e. We set $x_i=x_i^e+d_i$; therefore:

$$x_4^c=(x_4^e+d_4)(x_2^e+d_2)(x_1^e+d_1) =$$

$$x_4^e x_2^e x_1^e+d_4 x_2^e x_1^e+d_2 x_4^e x_1^e+d_1 x_4^e x_2^e+o(d^2), \text{ where}$$

$$d=\max(d_1,d_2,d_3,d_4,d_5,d_6,d_7).$$

Similar equations involving x_5^c, x_6^c, x_7^c can be written. Ignoring the $O(d^2)$ terms, the minimization problem to solve in order to determine the correct attenuations that are closest to the expert-given ones is:

$$
\begin{array}{lll}
 & d_4x_4^e x_1^e + d_2x_4^e x_1^e + d_1x_4^e x_2^e - (x_4^c - x_4^e x_2^e x_1^e) = 0 & (g_1(d_i)=0) \\
(1) & d_5x_2^e x_1^e + d_2x_5^e x_1^e + d_1x_5^e x_2^e - (x_5^c - x_5^e x_2^e x_1^e) = 0 & (g_2(d_i)=0) \\
 & d_6x_3^e x_1^e + d_3x_6^e x_1^e + d_1x_6^e x_3^e - (x_6^c - x_6^e x_3^e x_1^e) = 0 & (g_3(d_i)=0) \\
 & d_7x_3^e x_1^e + d_3x_7^e x_1^e + d_1x_7^e x_3^e - (x_7^c - x_7^e x_3^e x_1^e) = 0 & (g_4(d_i)=0)
\end{array}
$$

$$
\begin{array}{ll}
 & -x_1^e <= d_1 <= 1-x_1^e \\
(2) & -x_2^e <= d_2 <= 1-x_2^e \\
 & \cdots\cdots\cdots \\
 & -x_7^e <= d_7 <= 1-x_7^e
\end{array}
$$

$$\min z = d_1^2 + d_2^2 + \ldots + d_7^2.$$

Since we assume that the expert-given attenuations are close to the correct attenuations, it is likely that the inequality constraints labeled (2) are not <u>active</u>, i.e., the solution of the quadratic program with constraints (1) also satisfies constraints (2). In this case, the optimum solution can be found by using the method of Lagrange multipliers. The Lagrangian is

$$f(d_i) = z(d_i) + l_1(d_i) + \ldots + l_4 g_4(d_i).$$

Since the set identified by the equality constraints and $z(d_i)$ are both convex, the optimum solution is obtained by setting to zero the partial derivatives of f with respect to each of the d's and the l's. Since $f(d_i)$ is a quadratic function, this latter system is a system of linear equalities.

If the expert-given attenuations are not close enough to correct attenuations, the solution found as indicated above will not satisfy the inequality constraints (2). In this case, one or more of the inequalities constraints must be active. The optimum solution can be found by trying out combinations of active constraints. Of course, this procedure takes time exponential in the number of inequality constraints (and therefore in the number of attenuations.) However, a polynomial run-time algorithm exists for convex quadratic programming. The reader is referred to [Kozlov et al., 1979] for further details.

(End of example.)

It may be that the error introduced by linearizing is small enough that it is acceptable to compensate it by multiplying each leaf attenuations by the product of the expert-given attenuations on the branch ending at that leaf divided by the product of the correct attenuations. If this is not the case, that is the error introduced by linearizing is too big, the linearization process can be iterated.

Synthesis of attenuations: MIN/probabilistic sum case

Let a[p+]b mean the probabilistic sum of a and b. Consider trees composed of building blocks as in Figure 4.7.

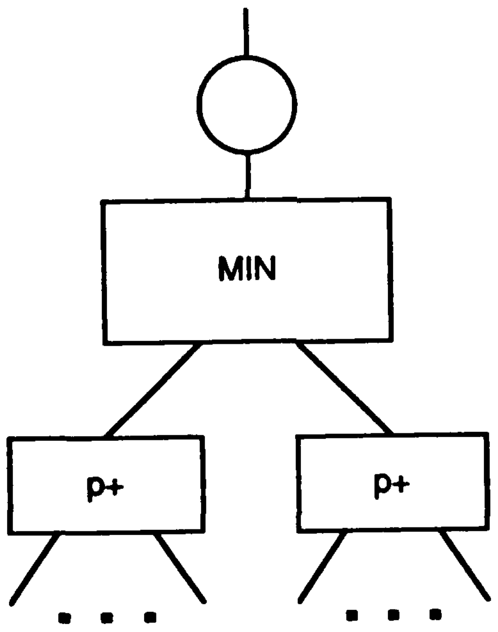


Figure 4.7 Subgraph for the probabilistic sum case

Let there be an oracle that can give the output part of a test when given the input part of that test, for any test. Attenuations are multiplicative between 0 and 1, and the minimum non-zero input allowed (min) is known. (This last requirement is based on the observation made for the MIN/MAX case that it makes it simple to determine whether the attenuations on a path can be synthesized, as will be made precise in the following lemma.) Note that, while in the MIN/MAX case only approximately half of the attenuations in a tree can be assigned values independently, no such result holds for the p+/MIN case.

Lemma 4.1 If a path is sensitizable, it is sensitizable by a test with min as path input, 0's as inputs to other branches entering the path at p+ boxes, 1's as inputs to other branches entering the path at MIN boxes.

Proof The path is sensitized with respect to the p+ boxes, because 0[p+]e = e, for all e >= 0. Since the minimum non-0 value at an input CF is min, the choice of min as input to the path to be sensitized provides for the smallest non-0 value in the path anywhere, in particular at the input to MIN boxes. On the other hand, because of monotonicity, 1's at the inputs corresponding to branches meeting the path we want to sensitize at MIN boxes maximize the CFs at these boxes. (In Figure 4.8, for example the leftmost path is sensitized.)

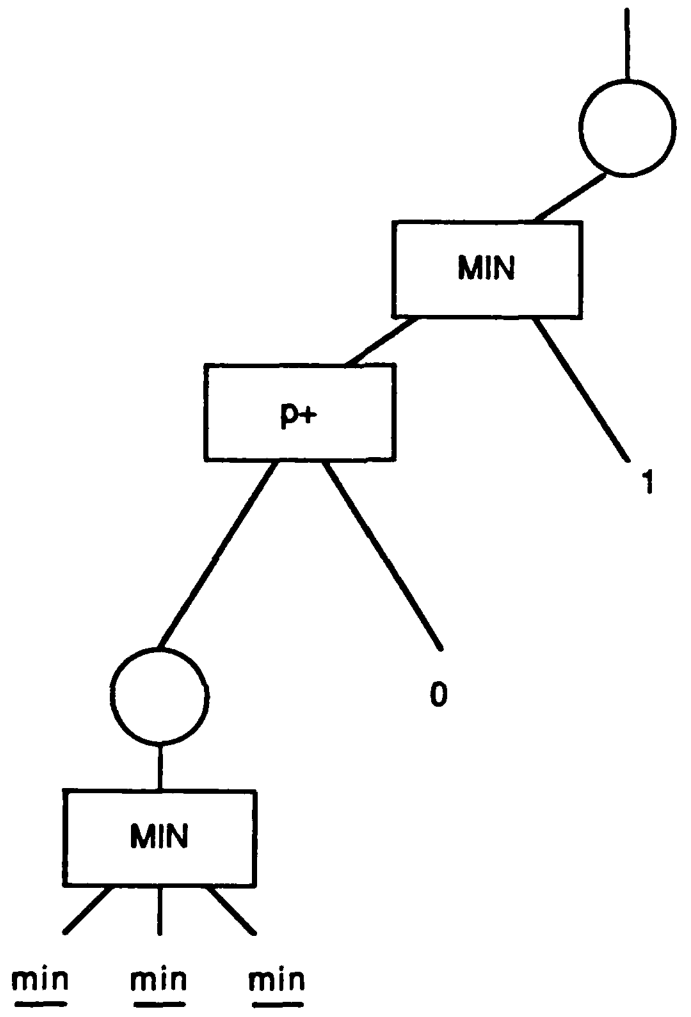


Figure 4.8 Sensitizing a path with probabilistic sum

(End of proof of lemma)

The proof of the Lemma describes a procedure to verify whether a path is sensitizable or not and to construct a test that sensitizes the path, if it exists, which will be used in the algorithm to synthesize attenuations in a MIN/p+ tree. Call this procedure method A. The following procedure is the other main building block of the algorithm to synthesize attenuations.

Method B Consider an attenuation, say a, in a tree built of the building blocks in Figure 4.7.

Assume that the following three tests exist: (a) a test that makes influential two of the paths through a p+ box just below a and such that the two CFs entering the p+ box are the same as those obtained in tests (b) and (c); (b) a test that makes influential the left one of the two paths mentioned for test (a); (c) a test that makes influential the right one of the two paths mentioned for test (a).

When the input part of test (b) is applied, the CF value exiting attenuation a is $k_1 = ab$, where b does not depend on a. When the input part of test (c) is applied, the CF value exiting attenuation a is $k_2 = ac$, where c does not depend on a. Similarly, when test (a) is applied, this value is $k_3 = a(b[p+]c)$. But $b[p+]c = b+c-bc = k_1/a+k_2/a-k_1k_2/(aa)$. Therefore, test (a) implies that $k_1+k_2-bc/a = k_3$. a can be computed from this.

Method A can be used to determine whether tests (a), (b), and (c) exist, and, if they do, to compute them.

(End of method B)

It is now possible to describe an algorithm that synthesizes all attenuations in a MIN/p+ tree.

Algorithm 4.1

```
If the test input "all 1's" produces output 0, set all
attenuations to 0 and stop.
For l=1 to depth of the tree, do
   for att=1 to number of attenuations at level l do
        if there are two paths through att that (a) go through
        the same p+ box, (b) are both influential, (c) the
        bundle obtained by joining them is influential, then
                synthesize attenuations using Method B
        else if there is one path through att which can be made
        influential, then
                set att to satisfy a test that synthesizes the
                path, using Method B to construct such a test
                set all attenuations in the subtree rooted at
                att to 1
        else (if no path through att can be made influential)
                set att and all attenuations in the subtree
                rooted at att to 1
```

Example 4.4

Consider the simple inference tree in Figure 4.9. In this example, it is assumed that no attenuations have value 0, i.e., all paths can be made influential. It is shown here how to compute a_4 and a_1.

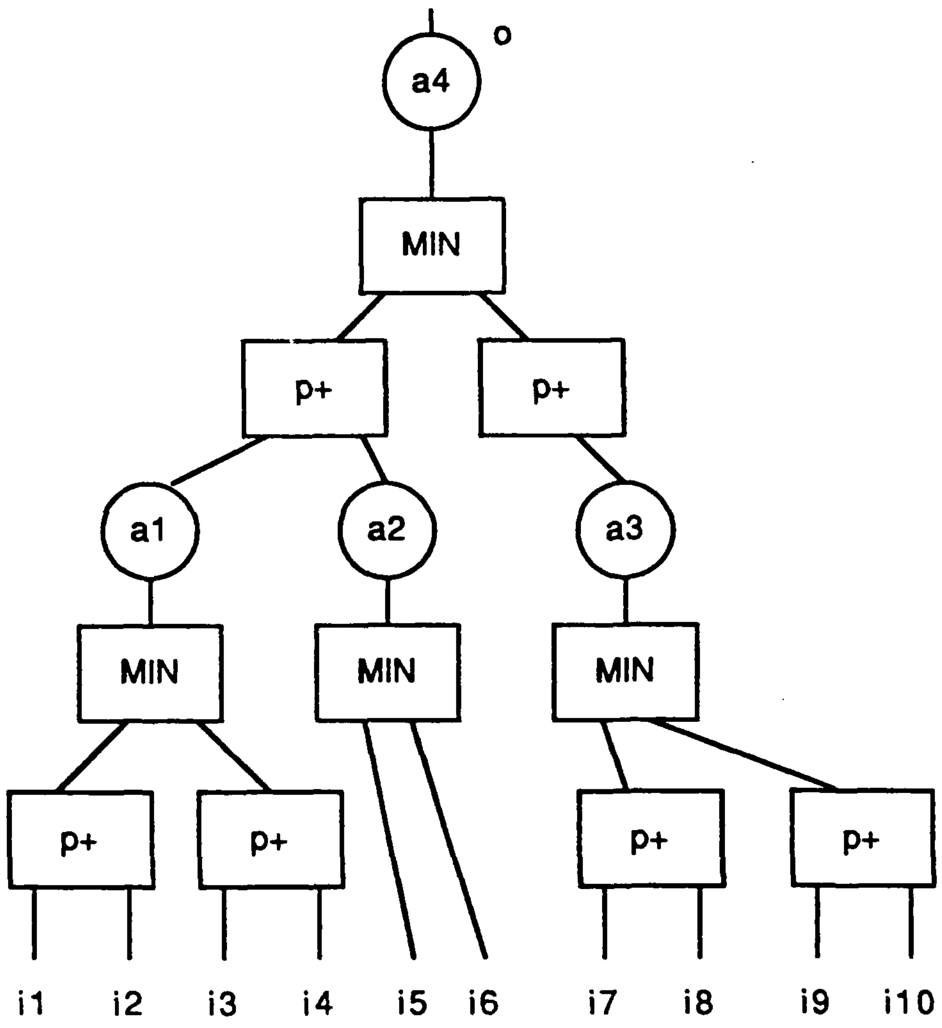


Figure 4.9 Example of use of algorithm 4.1

To synthesize a_4, use the following test inputs:

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}
(1)	min	0	1	1	0	0	1	1	1	1
	(makes path i_1 to o influential)									
(2)	0	0	0	0	min	1	1	1	1	1
	(makes path i_5 to o influential)									

(3) min 0 1 1 min 1 1 1 1 1
(makes the bundle composed of paths i_1 to o and i_5 to o influential) (Tests (1), (2), and (3) correspond to tests (b), (c), and (a) of Method B, respectively.)

The oracle gives values k_1, k_2, k_3 for the output corresponding to test inputs (1), (2), and (3), respectively. According to Method B, a_4 can be computed by solving the following system of inequalities:

$$a_4b=k_1$$
$$a_4c=k_2$$
$$k_1+k_2-bc/a_4=k_3$$

and therefore

$$a_4=[(k_1k_2)/(k_1+k_2-k_3)]^{-3}.$$

To synthesize a_1, use the following test inputs:

(1) min 0 1 1 0 0 1 1 1 1
(makes path i_1 to o influential)
(2) 0 min 1 1 0 0 1 1 1 1
(makes path i_2 to o influential)
(3) min min 1 1 0 0 1 1 1 1
(makes the bundle composed of paths i_1 to o and i_2 to o influential) (Tests (1), (2), and (3) correspond to tests (b), (c), and (a) of Method B, respectively.)

The oracle gives values k_4, k_5, k_6 for the output corresponding to test inputs (4), (5), and (6), respectively. According to Method B,

$$a_1 = 1/a_4[(k_4k_5)(k_4+k_5-k_6)]^{-3}.$$

(End of example)

CHAPTER FIVE

THE INCOMPLETE CASE

Introduction

In the incomplete case, we model sample cases by tests, defined in Section 3 of Chapter 2. The name, incomplete case, stems from the fact that the input/output function to be realized by the rule-based system is available only through samples, rather than completely through an oracle.

All results in this chapter concern trees. The first section introduces a restricted MIN/MAX tree topology and shows that attenuation synthesis is an NP-Complete problem even for very small strength alphabets. We then show that the problem remains NP-Hard for real strengths. This second result implies NP-Hardness of the first one. Instead of proving only the more general result, we prove the simpler case first and reference its proof when proving the more general case, for ease of exposition. The second section proves analogous results for trees where combinators are MIN functions and integrators are probabilistic sums. Since these are negative complexity results, they apply to more general cases, such as unrestricted trees and graphs. D.W. Loveland completed a proof that the unrestricted tree case is NP-Hard, before the analogous result in the restricted case, presented here, was proven.

Synthesis of attenuations: MIN/MAX case

This section contains the proof that the incomplete case on restricted MIN/MAX trees is NP-Complete.

First, the nature of the restriction.

We restrict the number of inputs to MIN boxes to be a constant but we do not restrict the fan-in to MAX boxes. This means that we can have any number of rules concluding the same hypothesis, but each rule can only have up to a fixed number of conjuncts in its premise. We restrict the depth of the inference tree, i.e. the length of the longest deduction, by a constant (fixed number, independent of the size of the net). We restrict MIN and MAX to be alternating.

These restrictions model certain observed properties of rule-based expert systems: they typically exhibit short inference chains, while their growth consists mostly of the addition of rules that conclude the same intermediate hypotheses already present. Moreover, we restrict the topology of the net to be tree.

For the purpose of the proof that follows, it is sufficient to consider nets with the topology described in Figure 5.1.

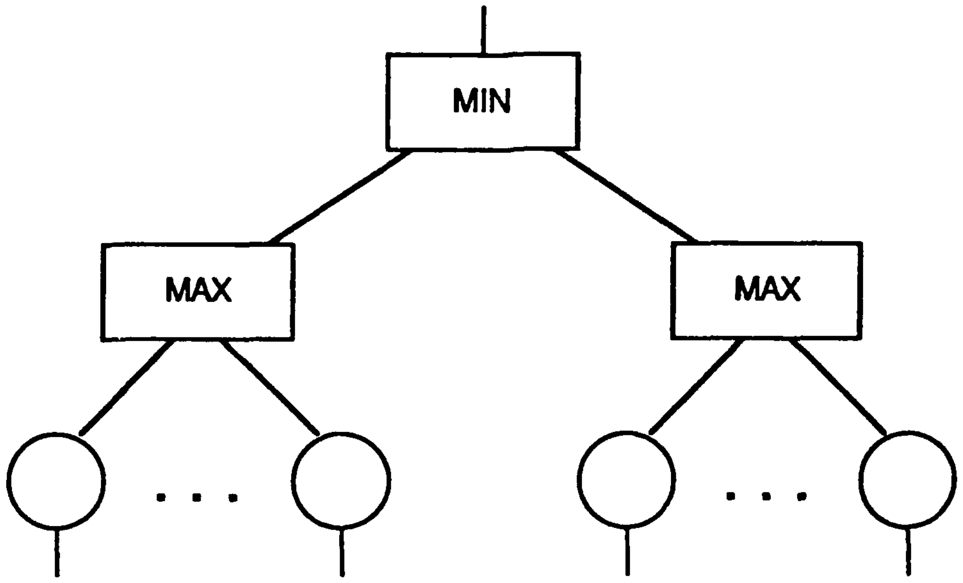


Figure 5.1 Restricted networks

We note that this topology does <u>not</u> have MIN's as leaf boxes. This is somewhat unsatisfactory, because one would expect to see the net composed of building blocks like that in Figure 5.2.

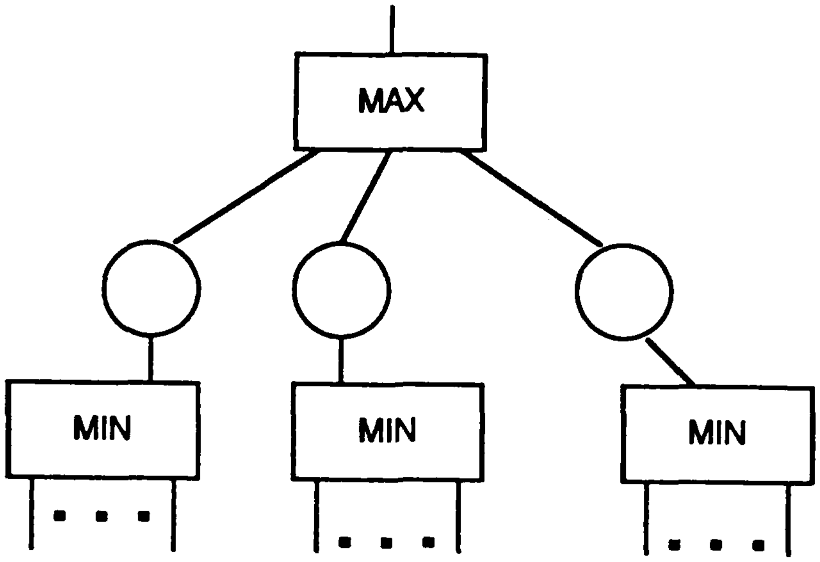


Figure 5.2 A natural building block

Each of the blocks in Figure 5.2 corresponds to rules concluding the same hypothesis. Still, the net we consider can be viewed as a shorthand for the more "natural" inference tree shown in Figure 5.3. To make this more precise, it will be shown that the inputs to the network in Figure 5.3 can be selected to simulate any desired assignment of inputs to the network in Figure 5.1. Number the inputs to the network in Figure 5.1, 1 through n, and the inputs to the network in Figure 5.3, 1 through 2n. The even inputs (2 through 2n) to the network in Figure 5.3 have value 1. Input j (where j is odd) to the network in Figure 5.3 has the same value as input floor(j/2) to the network in Figure 5.1. Clearly, the output of the network in Figure 5.3 is the same as the output of the network in Figure 5.1 for any input to the network in Figure 5.1 when the input to the network in Figure 5.3 is built according to the mapping just given.

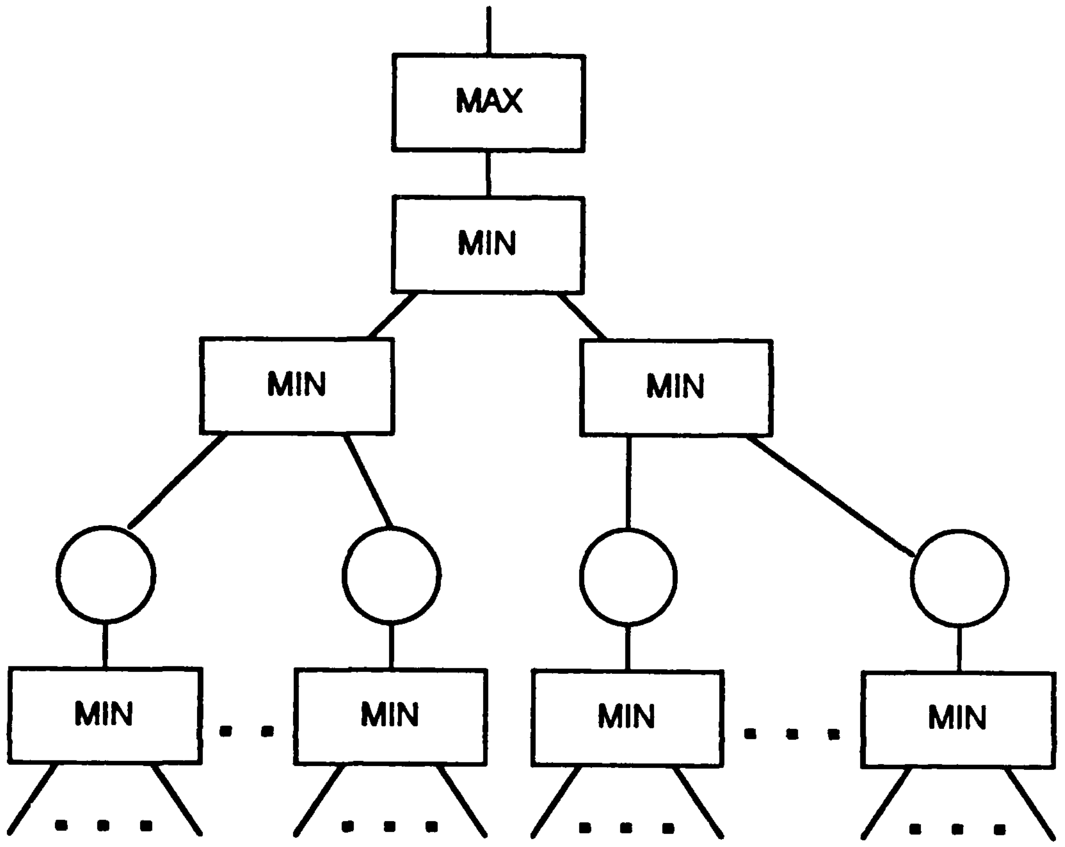


Figure 5.3 Network equivalent to that in Figure 5.1

The problem is formalized as follows.

Problem name. Restricted Attenuation Synthesis (RA).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with 0/1 values at the output of MIN boxes, bounded fan-in to MIN's, bounded depth; a set of tests.

Question. Is there an assignment of attenuations for which all tests are handled correctly?

Theorem 5.1 RA is NP-Complete.

Proof

Membership in NP is trivial. The non-deterministic program that solves the problem has a loop whose body is an assignment of 0 or 1 (non-deterministically) to each attenuation.

We transform Monotone 3-Conjunctive Normal Form Satisfiability (MSAT) [Garey and Johnson, 1979] to RA. The generic MSAT instance is a conjunctive normal form expression, where each clause contains only three negated or only three un-negated variables (e.g., $(x_1 v x_2 v x_3)$ & $(-x_1 v -x_4 v -x_5)$). The question is whether there is a satisfying truth assignment for the expression.

Given an expression E in monotone conjunctive normal form, the following algorithm will produce in time polynomial in the size of E an instance of RA such that the Question has answer yes if and only if E is satisfiable.

Let n be the number of distinct variables in E, m be the number of clauses in E. (n and m can be obtained in polynomial time from any "reasonable" encoding of E.)

The tree of (the) RA (instance) has three levels: a MIN box, two MAX boxes under it, attenuators under the MAX boxes. There are 2n+1 attenuators for each MAX box. For clarity we number them 1 through 4n+2. (The tree has size polynomial in n, and therefore in the size of E.) Figure 5.4 shows the tree.

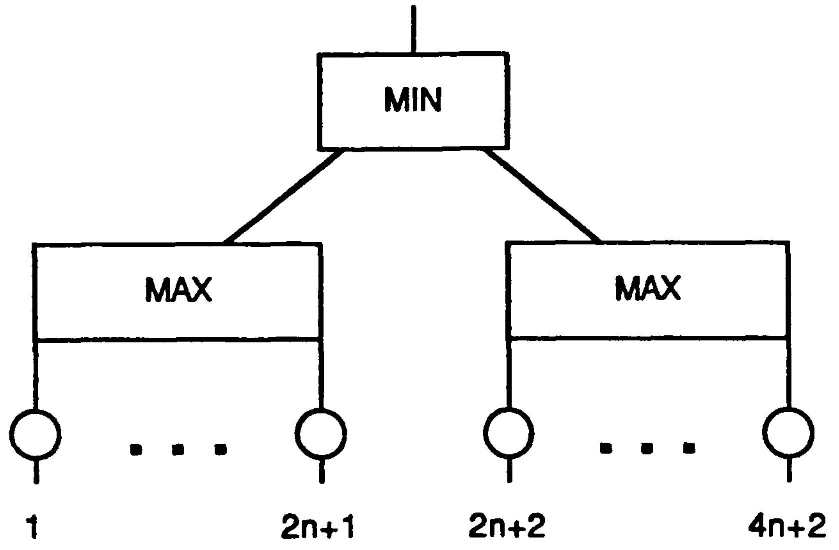


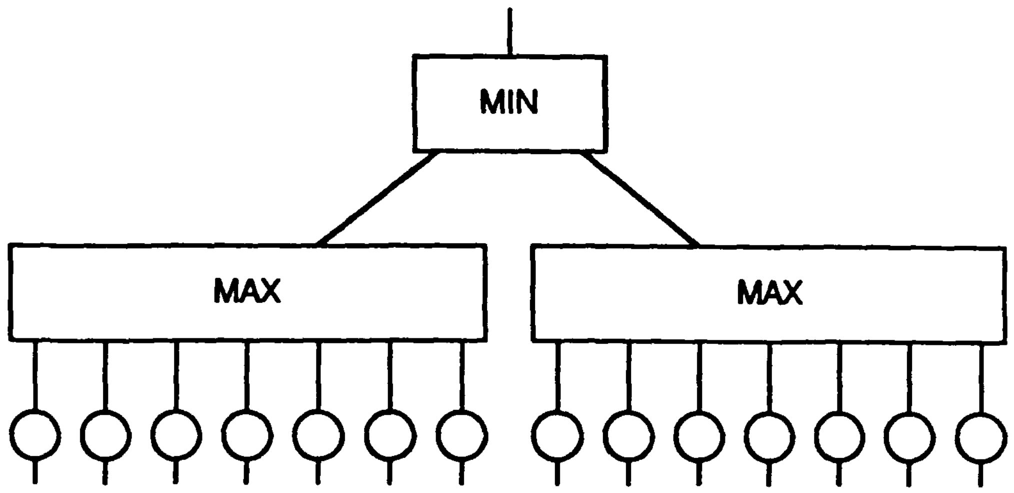
Figure 5.4. Tree of the generic RA instance

There are 2n+m tests for the RA instance. Here is how to build them. a) m <u>clause tests</u>. Each one of these tests corresponds to a clause. Each of these tests has output .6. Each of the inputs that is not explicitly specified in the following has value 0. If the clause is positive and contains the variables v_i, v_j, v_k, the test is built as follows: .6's for inputs i, j, k, .9 for input 4n+2. If the clause is negative and contains the variables v_i, v_j, v_k, .6's for inputs 2n+1+i, 2n+1+j, 2n+1+k, .9 for input 2n+1. (In fact, any two values a and b, such that $0 < a < b < 1$ will do.) Intuitively, one can say that the left MAX box is for positive clauses, while the right MAX box is for negative clauses.) (Clearly, these tests can be built in time polynomial in n and m.) b) 2n <u>variable tests</u>. Each pair of these corresponds to a variable. The first test of each pair has output 0. The second has output .6. Each of the inputs that is not explicitly specified has value 0. For variable v_i, the first test of each pair has a .6 for inputs i and 2n+i+1; the second test has .6's there and .9's in positions n+i and 2n+1+n+i. (Clearly, these tests can be built in time polynomial to n, the number of variables in E.)

An example of how MSAT instances are mapped into RA instances is given below.

Example 5.1

The MSAT instance is
$E = (x_1 v x_2 v x_3)$ & $(-x_1 v -x_2 v -x_3)$
$m=2, n=3$.
The corresponding RA instance is shown in Figure 5.5. T_1 and T_2 are the clause tests; T_3 and T_4 are the variable tests corresponding to x_1; T_5 and T_6 are the variable tests corresponding to x_2; T_7 and T_8 are the variable tests corresponding to x_3.



															output
T_1	.6	.6	.6	0	0	0	0	0	0	0	0	0	0	.9	.6
T_2	0	0	0	0	0	0	.9	.6	.6	.6	0	0	0	0	.6
T_3	.6	0	0	0	0	0	0	.6	0	0	0	0	0	0	0
T_4	.6	0	0	.9	0	0	0	.6	0	0	.9	0	0	0	.6
T_5	0	.6	0	0	0	0	0	0	.6	0	0	0	0	0	0
T_6	0	.6	0	0	.9	0	0	0	.6	0	0	.9	0	0	.6
T_7	0	0	.6	0	0	0	0	0	0	.6	0	0	0	0	0
T_8	0	0	.6	0	0	.9	0	0	0	.6	0	0	.9	0	.6

Figure 5.5 Instance of RA corresponding to $(x_1 v x_2 v x_3) \& (-x_1 v -x_2 v -x_3)$

(End of example)

Now, it must be shown that the instance of RA built according to the above algorithm is a yes-instance when E is satisfiable and a no-instance when E is not satisfiable.

Lemma 5.1. Consider the variable tests. Consider the tests corresponding to a generic variable v_i. The first test of each pair is satisfiable in isolation if and only if at least one of the attenuations in positions i and 2n+1+i is 0, because the output of the MIN box is 0 if and only if at least one of its inputs is 0. The second test is satisfied in isolation only if at least one of the attenuations in positions i and 2n+1+i is 1, because the output of MIN is .6 only if at least one of its inputs is .6. Therefore, exactly one of the two attenuations is 1 and the other is 0. (Note that the variable tests do not specify which of the attenuations is 1 and which one is 0.)

Lemma 5.2. Consider the clause tests. Consider the test corresponding to a generic clause, consisting of three literals, l_i, l_j, l_k. Assume that the literals are positive. The test is satisfied in isolation only if any of the attenuations in positions i, j, and k is set to 1, because the output of the MIN box is .6 only if at least one of its inputs is .6. The case for negative clauses is similar, but the attenuations are now in positions 2n+1+i, 2n+1+j, 2n+1+k.

(A) If E is satisfiable then RA is a yes-instance. In order for RA to be a yes-instance, there must be some assignment of attenuations such that all tests are "satisfied," i.e. the test inputs must produce the test output. Assume that each clause in E is satisfiable. Consider a generic clause first. Set to (1,0) the two attenuations in position i and 2n+1+i, respectively, if variable v_i in the clause is T in E's model; set the attenuations to (0,1) if variable v_i in the clause is F in E's model. Since E has a model, it can never be that such a pair of attenuations is mapped into (1,0) and (0,1) simultaneously. Therefore, by Lemma 1, the variable tests are satisfied. Now, note that each clause must be satisfied in order for E (in CNF) to be satisfied. Therefore, a positive clause will have a variable set to T in the model for E. This means that the corresponding test has .6 as output, because one of the .6's from the left MAX box will have the corresponding attenuation set to 1 and, being MINned with the .9 from the right MAX box, it will propagate to the output. Therefore, the test is satisfied. The case for negative clauses is analogous.

(B) If RA is a yes-instance, then E is satisfiable, i.e. E has a model. Given RA, build the model according to the following correspondence: if the attenuation in position i has value 1 and the attenuation in position 2n+1+i is 0, variable

v_i has value T; if these attenuations have values 0 and 1, variable v_i has value F. As already shown, (0,1) and (1,0) are the only two possible cases. Since E is in conjunctive normal form, E is satisfiable if (and only if) each clause is. Each clause is satisfiable if each clause test is, as shown in Lemma 2.

(End of proof.)

The proof that RA is NP-Complete can be extended to the case in which attenuations are allowed to be real numbers between 0 and 1, as shown below.

Problem name. Attenuation Synthesis (AS).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real weights at the output of MIN boxes, bounded fan-in to MIN's, bounded depth; a set of tests.

Question. Is there an assignment of attenuations for which all tests are handled correctly?

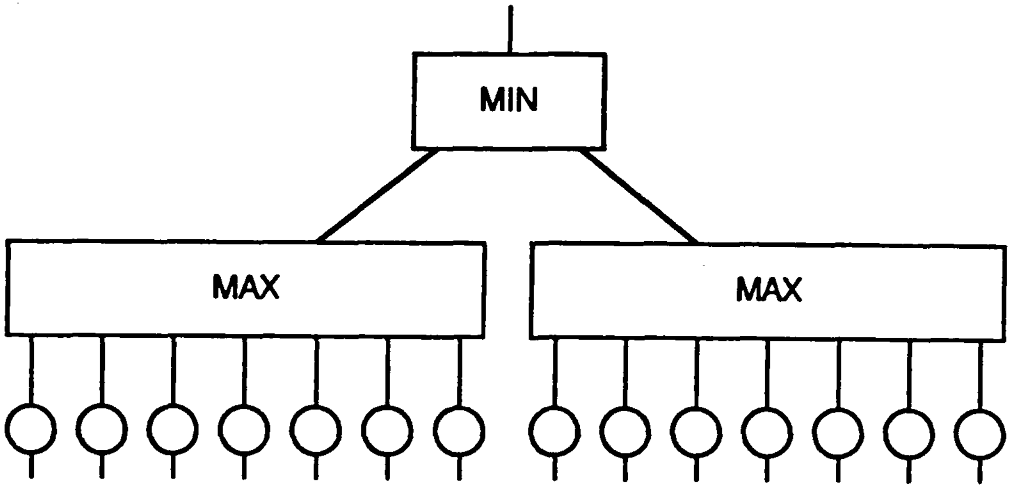
Theorem 5.2. AS is NP-Hard.

Proof.

The proof is analogous to the NP-Hardness part of the NP-Completeness proof for problem RA. In particular, the clause tests need not to be modified. The variable tests must be modified. A third test must be added for each variable. This test is equal to the second test, but .7's are in place of the .9's.

Example 5.2

The MSAT instance is
$E=(x_1 v x_2 v x_3)$ & $(-x_1 v -x_2 v -x_3)$
$m=2, n=3$.
Figure 5.6 shows the corresponding AS instance. T_1 and T_2 are the clause tests; T_3, T_4, and T_5 are the variable tests corresponding to x_1; T_6, T_7, and T_8 are the variable tests corresponding to x_2; T_9, T_{10}, and T_{11} are the variable tests corresponding to x_3.



															output
T_1	.6	.6	.6	0	0	0	0	0	0	0	0	0	0	.9	.6
T_2	0	0	0	0	0	0	.9	.6	.6	.6	0	0	0	0	.6
T_3	.6	0	0	0	0	0	0	.6	0	0	0	0	0	0	0
T_4	.6	0	0	.9	0	0	0	.6	0	0	.9	0	0	0	.6
T_5	.6	0	0	.7	0	0	0	.6	0	0	.7	0	0	0	.6
T_6	0	.6	0	0	0	0	0	0	.6	0	0	0	0	0	0
T_7	0	.6	0	0	.9	0	0	0	.6	0	0	.9	0	0	.6
T_8	0	.6	0	0	.7	0	0	0	.6	0	0	.7	0	0	.6
T_9	0	0	.6	0	0	0	0	0	0	.6	0	0	0	0	0
T_{10}	0	0	.6	0	0	.9	0	0	0	.6	0	0	.9	0	.6
T_{11}	0	0	.6	0	0	.7	0	0	0	.6	0	0	.7	0	.6

Figure 5.6 Instance of AS corresponding to $(x_1 v x_2 v x_3)\&(-x_1 v -x_2 v -x_3)$

Now, we show that the second and third tests are satisfied only if at least one of the attenuations in position i and 2n + 1 + i is 1. Clearly, the tests can be satisfied when either or both of the attenuations are 1. It will now be shown that the two tests in isolation cannot be satisfied when both attenuations are not 1. Assume that both attenuations are different from 1. In this case, the output .6 must propagate from inputs n+i or 2n+1+n+i, since all other inputs are attenuated to less than the desired value of the output (.6). Therefore, as shown in Figure 5.7, in order for the two tests to be satisfied in isolation, the attenuations in positions n+i, 2n + 1 + n + i, n+i, 2n + 1 + n + i must satisfy the constraints summarized in any of the four rows in the following table:

attenuations in position:

n+i	2n+1+n+i	n+i	2n+1+n+i
=6/9	>=6/9	=6/7	>=6/7
=6/9	>=6/9	>=6/7	=6/7
>=6/9	=6/9	=6/7	>=6/7
>=6/9	=6/9	>=6/7	=6/7

None of these four conjunctions is satisfiable. Therefore, the assumption does not hold, i.e., either the attenuation in position i or the attenuation in position 2n + 1 + i must be 1.

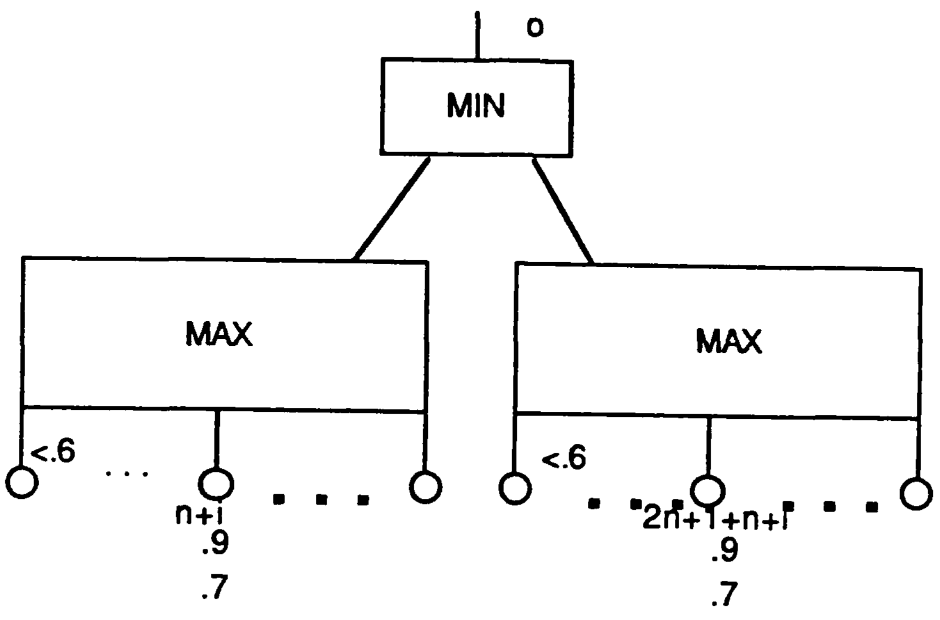


Figure 5.7 Constraints on attenuations

From this point on, the proof is totally analogous to the one for RA being NP-Hard.

(End of proof.)

There is a special case for which there is an efficient algorithm to synthesize attenuations. In this case, the network is as shown in Figure 5.8 below. Attenuations are multiplicative, with weights 0 and 1.

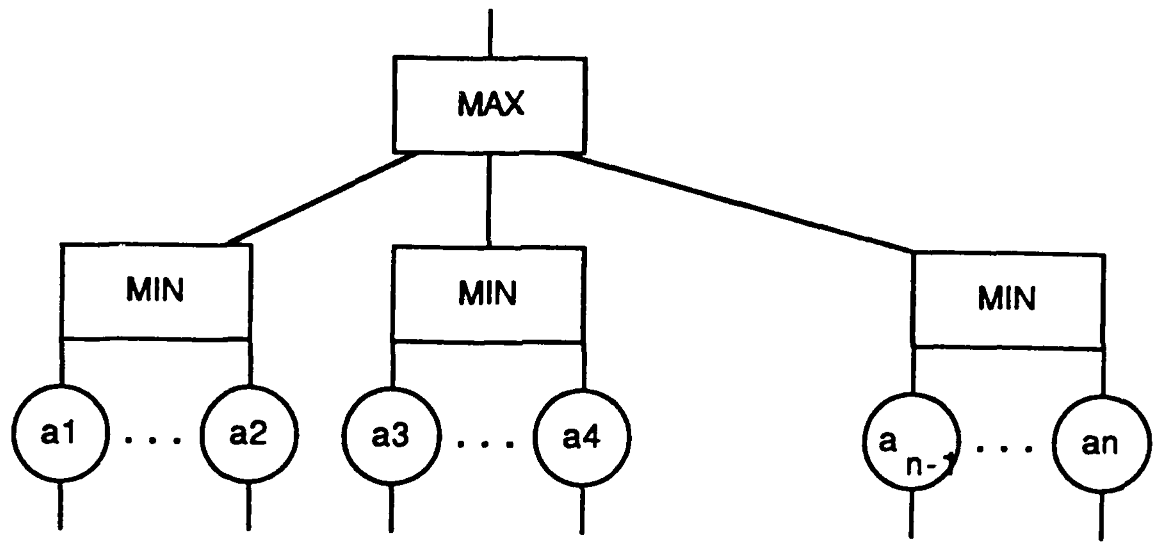


Figure 5.8. A simple inference tree.

The algorithm is the following. Consider each test in turn. Initialize all attenuations to 1. Take MIN of each pair of inputs; set the corresponding pair of attenuations to 0 if the MIN of the pair is greater than the output of the test.

By following a least commitment strategy, the algorithm finds an assignment of attenuations compatible with the tests if such an assignment exists.

<u>Incomplete case: MIN/Probabilistic sum</u>

It is shown that the incomplete case is NP-Hard with MIN, p+, even when trees of restricted depth and fan-in to MIN's are considered.

The restrictions introduced here are analogous to those described in "proof2." Note that the result is obtained without introducing MAX boxes.

We restrict the number of inputs to MIN boxes to be a constant, but we do <u>not</u> restrict the fan-in to p+ boxes. This means that we can have any number of rules concluding the same hypothesis, but each rule can only have up to a fixed number of conjuncts in its premise. The depth of the inference tree, i.e. the length of the longest deduction is restricted by a constant (fixed number, independent of the size of the net). We restrict MIN and p+ to alternate. It has been observed that many rule-based expert systems exhibit short inference chains and grow by the addition of rules that conclude the same intermediate hypotheses already present. We also restrict the topology of the net to be tree.

For the purpose of the proof that follows, it is sufficient to consider the topology in Figure 5.9, where a indicates attenuation. (The attenuation feeding directly into the MIN box is to be considered a shorthand for a degenerate, 1-input p+ box with attenuated input.)

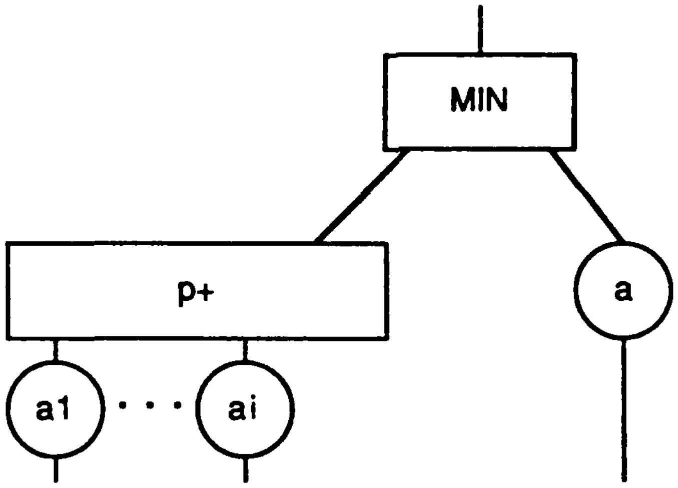


Figure 5.9 The generic RAP instance

The problem is formalized as follows.

Problem name. Restricted Attenuation Synthesis (MIN/p+ case) (RAP).

Problem instance. A tree with alternating MIN and p+ boxes, multiplicative attenuators with 0/1 values at the input of p+ boxes, bounded fan-in to MIN's, bounded depth; a set of tests.

Question. Is there an assignment of attenuations for which all tests are handled correctly?

Theorem 5.3 RAP is NP-Complete.

Proof

Membership in NP is trivial. The non-deterministic program that solves the problem has a loop whose body is an assignment of 0 or 1 (non deterministically) to each attenuation.

We transform One in Three Satisfiability (OTS) [Garey and Johnson, 1979, p.259] to RAP. We use the variant in which no clause in the expression contains a negative literal. The generic instance of this problem is an expression in conjunctive normal form, with no negative literal. The question is whether there is a model for the expression such that each clause has exactly one true literal.

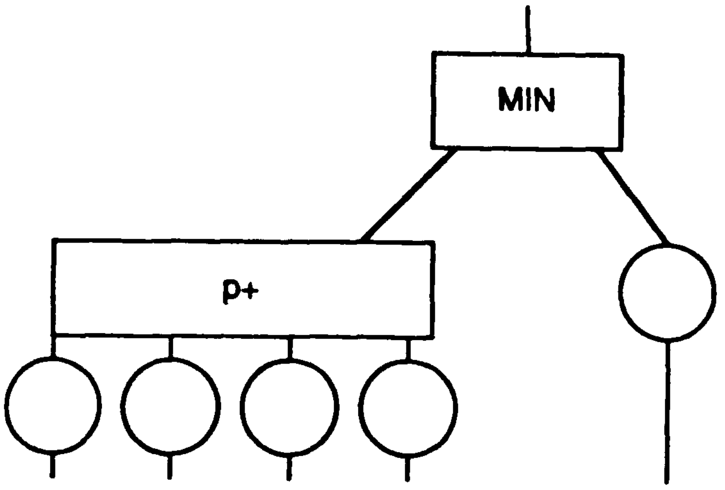
Given an Expression E in monotone 3-conjunctive normal form, the following algorithm will produce in time polynomial in the size of E an instance of RAP such that the Question has answer yes if and only if E has a model in which only one literal per clause is true.

Let n be the number of distinct literal in E, m be the number of clauses in E. (n and m can be obtained in polynomial time from any "reasonable" encoding of E.)

The p+ box in the instance of RAP has n inputs. There are 2m tests: two tests correspond to each clause in E. Let a generic clause contain the variables v_i, v_j, v_k. The first test for this clause has values .6 corresponding to inputs i, j, k, value .7 corresponding to the lone attenuated input to the MIN box, all other input values set to 0, and output value .6. The second test for this clause has values .7 corresponding to inputs i, j, k, value .8 corresponding to the lone attenuated input to the MIN box, all other input values set to 0, and output value .7.

Example 5.3

Figure 5.10 shows the instance of RAP corresponding to
$E=(x_1 v x_2 v x_3)$ & $(x_2 v x_3 v x_4)$
n=4, m=2.
In the example, T_1 and T_2 correspond to the first clause in E, while T_3 and T_4 correspond to the second clause in E.



						output
T_1	.6	.6	.6	0	.7	.6
T_2	.7	.7	.7	0	.8	.7
T_3	0	.6	.6	.6	.7	.6
T_4	0	.7	.7	.7	.8	.7

Figure 5.10 Instance of RAP corresponding to $(x_1 v x_2 v x_3) \& (x_1 v x_2 v x_4)$

(End of example)

It will be shown that this instance of RAP is a yes-instance if and only if the corresponding instance of OTS is a yes-instance.

The if part is simple. If a variable in the model for E is T, set the corresponding attenuation to 1; otherwise (with the exception of the attenuation input to the MIN, which is also set to 1), set attenuations to 0. Since this insures that exactly one of the attenuations input to p+ corresponding to non-0 input test values is 1 for each test, the input test value propagates unchanged to the output and therefore each test is satisfied.

Only if part. It will be shown that if RAP is a yes-instance, then necessarily OTS is a yes-instance. Assume that RAP is a yes-instance. It will be shown that in order for RAP to be a yes-instance it must be that exactly one of the attenuations corresponding to the p+ box inputs is 1 for each test. By assigning T to the variable corresponding to this unique attenuation, we obtain a model for E which satisfies the "one in three" condition. Consider a generic pair of tests corresponding to a clause in E. We show, by algebraic manipulation, that this pair is satisfied if and only if exactly one of the three attenuations corresponding to the tests is 1. Call the attenuations x, y, and z. The pair of tests is satisfied if and only if the following system has a solution:

$$.6x\ [p+]\ .6y\ [p+]\ .6z = .6$$
$$.7x\ [p+]\ .7y\ [p+]\ .7z = .7$$

$$.6x + .6y - .36xy + .6z - .36xz - .36yz + .216xyz = .6$$
$$.7x + .7y - .49xy + .7z - .49xz - .49yz + .343xyz = .7$$

$$x + y - .6xy + z - .6xz - .6yz + .36xyz = 1$$
$$x + y - .7xy + z - .7xz - .7yz + .49xyz = 1$$

If any two of x, y, z are equal to 0 and the other is 1, this is satisfied. Otherwise, we should have (by subtracting the second from the first equation, side by side):

$$xy + xz + yz = .13xyz$$

Let $z > 0$ (A totally analogous argument holds for $x > 0$ or $y > 0$.)

$$xy/z + x + y = .13xy$$

$$.13xy > x + y,$$

impossible for $0 <= x <= 1$, $0 <= y <= 1$.

(End of proof.)

<u>Observation</u> The second part of the preceding proof, unchanged, shows that attenuation synthesis is NP-Hard when attenuations are allowed to vary between 0 and 1, rather than being restricted to 0 and 1.

CHAPTER SIX

CHAINS

Introduction

The chain is the simplest topology for an inference net. (A chain of n rules is a rule base that contains only n rules of the form IF P_i THEN P_{i+1} WITH ATTENUATION A_i, $0 < i < n+1$.) Synthesis of attenuations in a chain is a trivial matter if the attenuations are closed under composition and the identity attenuation is one of the allowable ones. Here, we consider the case in which attenuations are not closed under composition. To make the problem more realistic (and more challenging), it is assumed that attenuations are monotone non-decreasing functions mapping values into smaller values.

Attenuations that are not closed under composition do not allow a simple form of learning, called chunking (Rosenbloom et al., [1985]) that consists of substituting a chain of rules with a single rule; it may be useful to do this for reasons of efficiency, for example.

The result that is presented next provides another reason to prefer attenuations that are closed under composition.

Complete and incomplete cases

The complete and the incomplete cases are hard when attenuations are not closed under composition.

In this case, one can model the problem as follows.

The attenuations are functions from CF's to CF's. The test cases are a collection of (input vector, output) pairs. To simplify matters, we consider a chain of rules only, i.e. an expert system with a single input feature and a single output feature. (By feature, here we mean an attribute-value pair.) We also assume that all functions are finite functions. Suppose that we have a chain of rules $r_1, r_2, \ldots, r_m$. Suppose that the CF values range over the set A. A test set is the (possibly incompete) specification of a finite function h:A->A.

(This way, we cover both the complete and the incomplete case.) The attenuations are functions f_i:A->A, and the set of all attenuations is called F.

Synthesizing attenuations for the chain of length m is tantamount to finding a composition of functions from F of length m that is identical to h (possibly, just as far as h is specified by the test set; in this case, it is indifferent what the composition of f's is outside the specification).

This problem is related to finite function composition, a known NP-hard problem. In fact, this problem is PSPACE-Complete if there is no restriction on the length of the composition [Kozen, 1977]; it is NP-Complete if the length of the composition is restricted to be K or less and K is expressed in unary [Garey and Johnson, 1979]

We now show that the problem is NP-Complete even if the following additional restriction is added: "all the functions are monotone." The statement of the problem and a motivation of the model used follow.

Name: Monotone function composition (MFC).

Instance: A type declaration of a vector of subrange of integers (type T), a family F of Pascal procedures that take as input a vector of type T and return a vector of type T, such that the output vector is no greater (componentwise) than the input vector (i.e., f_i in F is a monotone function from T to T), a special monotone procedure h from T to T, and an integer K.

Question: Can h be obtained by composing procedures in F in such a way that the length of the composition is K? (i.e., is there a sequence of K indices such that h = f_{i1} o f_{i2} o ... o f_{ik} ?).

<u>Theorem</u>

MFC is NP-Hard.

<u>Note on encoding being concise</u>

The customary encoding of finite functions is given by the list of ordered pairs that is the graph of the finite functions. However, the encoding used in this proof is more concise, since the functions are encoded as Pascal procedures. The customary encoding seems extravagantly long in this case: it would seem that attenuations would be described, whenever possible, by a procedure to compute their output, given their input, rather than by a list of (input, output) pairs. A similarly concise encoding is used by Even and Goldreich [1981] for their proof that determining whether a given permutation can be obtained as the composition of permutations from a given set is NP-Complete.

Proof

Exact cover by three sets (X3C) is transformable to MFC. Here is the definition of the X3C problem, after [Garey and Johnson, 1979]. Instance: A finite set X with $|X|=3q$ (q an integer) and a collection C of 3-element subsets of X. Question: Does C contain an exact cover for X, i.e., a subset C′ of C such that every element of X occurs in exactly one member of C′?

An algorithm is given to construct an instance of MFC from a given instance of X3C in polynomial time. This instance of MFC has the property that the instance of X3C is solvable if and only if the instance of MFC is.

A generic instance of X3C is $U = \{e_i\}$, $i=1...3n$, $S = \{S_j\}$, $j = 1...m$, where $S_j = (e_{j1}, e_{j2}, e_{j3})$.

Algorithm

The integer K is n. The vector declaration is:

```
type a= array[1..3n] of 0..2;
(Note that 3n is a constant here!)
```

The procedure h is:

```
procedure h(inp:a, var out:a); /* "predecessor function" */
begin
   for i:=1 to 3n do
      if inp[i]<>0 then out[i] := out[i] - 1
                   else out[i] := in[i] end;
```

Examples

```
(3n = 6)
inp: [222222] [222211] [111011]
out: [111111] [111100] [000000]
```

(End of examples.)

For each S_j in S, construct the following procedure f_{Sj}

```
procedure fSj(inp:a, var out:a); begin
   if inp[j1]<>0 then out[j1] := in[j1] - 1
                 else out[j1] := in[j1];
   if inp[j2]<>0 then out[j2] := in[j2] - 1
                 else out[j2] := in[j2];
```

```
      if inp[j3]<>0 then out[j3] := in[j3] - 1
                    else out[j3] := in[j3]
   end;
```

Examples

($3n = 6$, $S_i = (e_2, e_4, e_5)$)
inp: [222222] [212112] [122121]
out: [212112] [202002] [112011].

(End of examples.)

(End of algorithm.)

Clearly this instance of MFC is built in time polynomial in n and m, because h has size linear in the size of the number 3n, the fS_i's have size linear in the size of the numbers j_1, j_2, j_3, the number of the fS_i's is bounded by m, and the size of the type declaration is bounded by 3n.

Now I show that if (the) MFC (instance) has a solution, then (the) X3C (instance) does too.

If MFC has a solution, then the composition f_{i1} o f_{i2} o ... o f_{ik} must be such that the vector [22...2] is mapped to [11...1], because of the definition of h.

But this can be true if and only if each entry in the vector is decremented by exactly one f_i.

Therefore, the subcollection of triples $\{S_{i1}, S_{i2}, \ldots, S_{ik}\}$ is an exact cover for U. (Note that the size of this cover is n, as it should, because K=n.)

Finally, I show that if X3C has a solution, then MFC does too.

If X3C has a solution, then there is a collection of triples $\{S_{i1}, S_{i2}, \ldots, S_{in}\}$ which is an exact cover for U.

Consider the function f_{Si1} o f_{Si2} o ... o f_{Sin}. First, note that K=n, and therefore the composition is of the right length.

Then, note that this function "fuzzy-decrements" each entry in each vector of type a exactly once.

(End of theorem.)

The "functions" f_i and h map vectors into vectors. The functions we are interested in map CF-values to CF-values (e.g., integers to integers). Of course, the vectors can be mapped into integers and viceversa. For example, read [222222] as $(222222)_3$. Therefore, if MFC is NP-hard, so is the problem of composing finite monotone functions of integers.

This handles the complete specification case.

Now, consider the incomplete specification case. If the specification given by the test cases maps only p out of q possible CF values, just consider the set of functions F' from $A|_p$ to $A|_{p'}$, where $A|_p$ is the set of p CF values that appear in input part of the specification, and $A|_{p'}$ is the set of p' CF values that appear in the output part of the specification. If $\min(|p|,|p'|) = O(|A|)$, the proof given for the complete case carries through with minor modifications for the incomplete case as well.

CHAPTER SEVEN

GRAPHS

Introduction

Graphs are the most general topologies for inference nets. Only acyclic graphs are considered here. One would expect synthesis of attenuations in graphs to be be difficult: this is indeed the case. Therefore, we consider a problem which is simpler (in a suitable and informally appealing sense) than those considered in ~~the previous chapters~~: the determination of one attenuation, in the complete case.

The results we obtain in this chapter have counterparts in the theory of switching functions. In particular, we find a fast polynomial-time algorithm to verify whether a fault on an input line in a monotone circuit is undetectable. To the best of our knowledge, this algorithm is faster than any known one.

Attenuation synthesis in MIN/MAX graphs with 0/1 weights

In this chapter, we consider the determination of one attenuation in a graph. A net is given with all attenuator values specified. A function that the net is supposed to realize is also given. However, the given attenuator values are not necessarily correct, i.e., the mapping between net inputs and net outputs is not necessarily that specified in the function. We only consider the problem of detecting whether the value of a specific attenuator is correct or not. In other words, it is assumed that one of these two cases holds: (1) the net with the given attenuator values realizes the function given in the problem statement; (2) the net with one attenuation value changed, for a given attenuation, implements the given function. The problems described in this chapter all deal with identifying whether case (1) or (2) holds.

First, assume that the following parameter choices hold: (a) MIN, MAX combinators and integrators; (b) 0/1 weights and CF's; (c) NOT or identity predicate functions.

Problem name. Detection of incorrect weight, general case

(DG).

Problem instance. An inference net with parameters as described above. A function to be realized by the inference net. A marker for a special attenuator in the net. All attenuators in the net are given value 1.

Question. Is there a test that detects whether the marked attenuator should have value 0?

The second problem is like DG, except that the marked attenuators are restricted to be input attenuators.

Problem name. Detection of incorrect weight, input case (DI).

Problem instance. An inference net with parameters as described above. A function to be realized by the inference net. A marker for a special input attenuator in the net. All attenuators in the net are given value 1.

Question. Is there a test that detects whether the marked (input) attenuator should have value 1?

Comment 1 The question asked in DG and DI can be rephrased as "Is the net redundant with respect to the CF value attenuated by the marked weight?"

Comment 2 The problems are equivalent to that of detecting a stuck-at-0 fault on the line (input, in the case of DI) to a Boolean circuit with the same topology as the inference net. Example 7.1 illustrates this correspondence.

Results Problems DG and DI are NP-Complete. DI is NP-Complete by a slight modification of Theorem 3.2 by Ibarra and Sahni [1975]; since DI is a special case of DG, and DG is in NP, DG is also NP-Complete.

Consider a net with the following parameter choices: (a) MIN, MAX combinators and integrators; (b) 0/1 weights and CF's; (c) identity predicate function.

Problem name. Detection of one incorrect weight, general monotone case (DGM).

Problem instance. As for DG.

Question. As for DG.

Problem name. Detection of one incorrect weight, input monotone case (DIM).

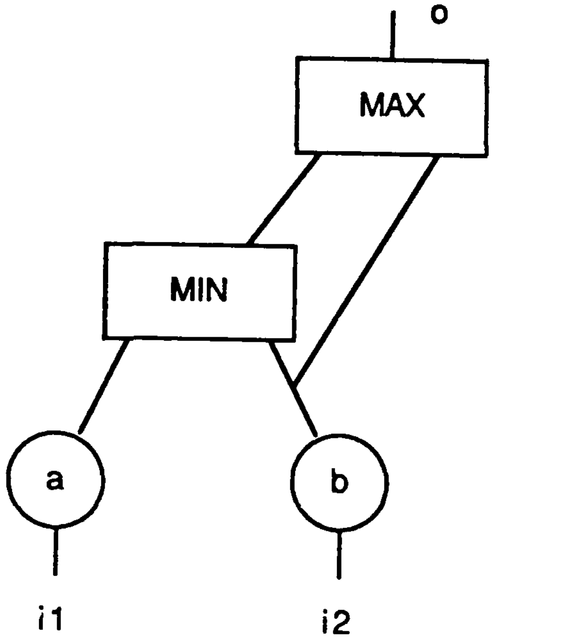
Problem instance. As for DI.

Question. As for DI.

Comment These problems are equivalent to that of detecting a s-a-0 fault on a line (input line in the case of DIM) of a monotone Boolean circuit.

Example 7.1

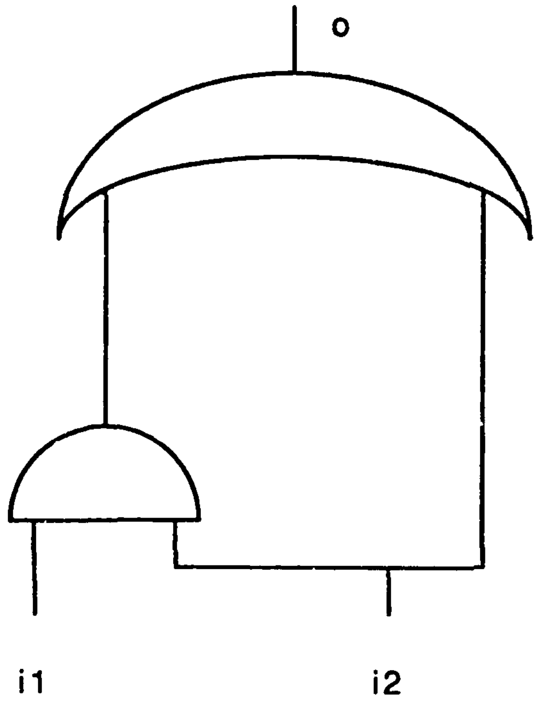
In this example (Figure 7.1), we illustrate the correspondence between checking the value of an attenuation in an inference net and detecting whether a line or a gate input in a monotone Boolean functions is stuck at 0.



a=1.

should b be 0?

(a)



Is i_2 stuck at 0?

(b)

Figure 7.1 Correspondence between certain inference nets and Boolean networks

(End of example)

One could therefore consider using a standard fault-detection technique to solve problem DGM. The D-Algorithm is such a technique [Breuer and Friedman, 1976]. Unfortunately, the D-Algorithm, which has exponential worst-time complexity, also has exponential worst-time complexity for monotonic networks. Moreover, Fujiwara and Toida [1982] show that problem DGM is NP-Complete. However, a modification of the D-Algorithm is shown to solve the fault-detection problem in polynomial worst-case time complexity when there are only a "small number" of reconvergent paths in [Fujiwara and Toida, 1982]. This would seem to be a practically important case for inference networks that are almost trees. In particular, Fujiwara and Toida's result shows that problem DGM, restricted to graphs that are almost trees, can be solved in polynomial time providing a bridge between the simple tree case and the NP-Complete problem DGM. In the remainder of this section, problem T_1, restricted to monotone networks, will be considered.

Let f be the function realized by the net with weight value 1. It is a monotone Boolean function.

A test $\underline{x}$ will distinguish the case "weight = 0" from the case "weight = 1" on the i-th input if and only if
$f_i(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n) = f_i(1) <>$
$f_i(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n) = f_i(0)$.
Since the function f is monotonic, in order for $f_i(1) <> f_i(0)$, it must be that $f_i(0)=0$ and $f_i(1)=1$.

Therefore, to find a test, one must find an input vector $\underline{x}$ such that:
(a) $\underline{x}$ has a 1 in the i-th position;
(b) $f(\underline{x}) = 1$;
(c) $f(\underline{x}^i{}_0) = 0$, where $\underline{x}^i{}_0$ is the vector equal to $\underline{x}$ in all positions, but with a 0 in the i-th position.

Definition 7.1 Let $\underline{x}$ and $\underline{y}$ be vectors. $\underline{x}$ is said to be covered by $\underline{y}$ (indicated by $\underline{x} <= \underline{y}$) if each component of $\underline{x}$ is less than or equal to the corresponding component of $\underline{y}$. Call this relation is covered by and the inverse relation covers. We also say that $\underline{x}$ is a descendent of $\underline{y}$ and that $\underline{y}$ is an ancestor of $\underline{x}$. This relation is a partial order. If the numbers are restricted to 0 and 1, the vectors form a Boolean lattice. $\underline{x}$ is an immediate descendent of $\underline{y}$ if $\underline{x}$ is a descendent of $\underline{y}$ and they differ in only one component, i.e., $\underline{x}$ has a 0 where $\underline{y}$ has a 1 and they are otherwise equal. The universal upper bound (u.u.b.) of this lattice is the vector of all 1's; the universal lower bound (u.l.b.) of this lattice is the vector of all 0's. A Boolean lattice is isomorphic to the field of all the subsets of its join-irreducible elements (points). This formalizes the correspondence between subsets and the bit vectors representing them.

In the remainder of this section, only vectors made of zeros and ones will be considered.

<u>Fact</u> Assume that $f(\underline{x}_i) = 0$, where $\underline{x}_i$ is the vector with zeros in each component, except for component i. Assume that f(u.u.b.) = 1. Because of monotonicity, a test for input i can be found by following <u>any</u> chain of bit vectors covering $f(\underline{x}_p)$. Such an algorithm takes O(n) function evaluations (i.e., queries to the perfect expert), where n is the number of inputs to the inference net. In the remainder of this section, it will be shown how to find the test in O(logn) function evaluations.

<u>Definition 7.2</u> A vector $\underline{x}$ is a <u>critical vector</u> if
(a) $f(\underline{x}) = 1$;
(b) $f(\underline{x}^i_0) = 0$ for all i.
Let $\underline{x}$ be a critical vector that covers $\underline{x}_p$.

Then $\underline{x}$ is a test for input i, because
(a) $\underline{x}$ has a 1 in the i-th position, since it covers $\underline{x}_p$;
(b) $f(\underline{x}) = 1$, since it is a critical vector;
(c) $f(\underline{x}^i_0) = 0$, since $\underline{x}$ is a critical vector.

<u>Definition 7.3</u> A vector $\underline{x}$ for which $f(\underline{x}) = 1$ ($f(\underline{x}) = 0$) is a <u>1-vector</u> (<u>0-vector</u>).

<u>Example 7.2</u>

For the four-input monotone function represented in Figure 7.2, vectors 1110 and 1010 are tests for input 1 stuck-at-0. Vector 1110 is a test, because f(1110) = 1 and f(0110) = 0; vector 1010 is a test, because f(1010) = 1 and f(0010) = 0. Vector 1010 is also a critical vector, because f(1010) = 1, f(1000) = f(0010) = 0. In Figure 7.2, 1-vectors are indicated by (1), 0-vectors by (0).

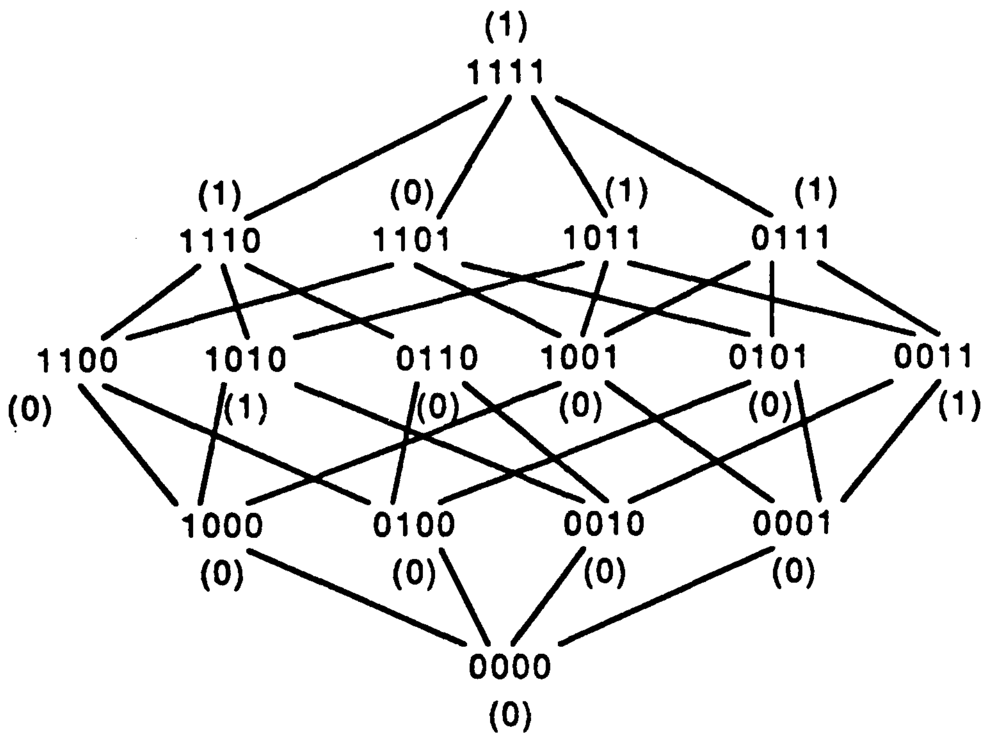


Figure 7.2 A four-input monotone Boolean function

(End of example)

<u>Theorem 7.1</u> If there is a test for an input attenuation i, then there is a critical vector which is a test for the same attenuation.

<u>Proof</u>

Assume f(u.l.b.) = 0, f(u.u.b) = 1. (If this condition does not hold, then there is no test, and the theorem is trivially true.)

Let $\underline{x}_p$ be the vector with 0's anywhere, but position i, where there is a 1. Any test $\underline{x}_t$ is such that $\underline{x}_p <= \underline{x}_t$.

If $\underline{x}_p$ is a 1-vector, then $\underline{x}_p$ is a test and must be a critical vector, because f(u.l.b) = 0.

Assume $f(\underline{x}_p) = 0$. (If there is no test, the theorem trivially holds.)

Let $\mathbf{x}$ be a test for the i-th input attenuation. Let q be the number of i's in $\mathbf{x}$.

The least number principle allows to conclude that there is a test with the least number of 1's. (A "lowest-level test," informally.)

Let $\mathbf{x}_1$ be such a test.

Now, we show that, if $\mathbf{x}_1$ is not a critical vector, then $\mathbf{x}_1$ is not a lowest-level test vector, thereby establishing the desired result by contradiction.

Assume that $\mathbf{x}_1$ is not a critical vector.

Since $\mathbf{x}_1$ is a test, its immediate descendent with a "0" in the i-th position, $\mathbf{x}_{1i/0}$, is a 0-vector.

Since $\mathbf{x}_1$ is not a critical vector, it must have an immediate descendent which is a 1-vector. This descendent must be a vector with "1" in the i-th position, because $\mathbf{x}_{1i/0}$ is a 0-vector. Therefore, this descendent covers $\mathbf{x}_p$.

Call the descendent $\mathbf{x}_{1d}$.

$\mathbf{x}_{1d}$ is a test, because: (a) $\mathbf{x}_{1d}$ covers $\mathbf{x}_p$; (b) $\mathbf{x}_{1d}$ is a 1-vector; (c) $\mathbf{x}_{1di/0}$ is a 0-vector, since $\mathbf{x}_{1di/0} < \mathbf{x}_{1i/0}$, $\mathbf{x}_{1i/0}$ is a 0-vector, and f is monotone.

Therefore, $\mathbf{x}_1$ is not a lowest-level vector which is a test, because there is a descendent of it, $\mathbf{x}_{1d}$, which is also a test. We have a contradiction.

Therefore, $\mathbf{x}_1$ is a critical vector.

(End of theorem.)

The previous fact and theorem allow us to conclude that in order to find a test for input attenuation i (or input i stuck-at fault), it is sufficient to find a critical vector that covers the vector with 0's everywhere except for a 1 in position i. A simple modification of the <u>critical set algorithm</u> [Loveland, 1982] will do the job. Here it is.

<u>Algorithm 7.1</u>--Find a critical vector covering $\mathbf{x}_p$.

```
1. C <- $\mathbf{x}_p$
   A <- u.u.b
   R <- u.l.b.  2. Find two vectors $A_1$ and $A_2$ s.t.
   (a) $A_1$ and $A_2$ together cover all points covered by A
   {correctness}
   (b) each of $A_1$ and $A_2$ cover fewer points than A does
   {termination}
   (c) $A_1$ and $A_2$ cover the same number of points {efficiency}
3. If $(C \cup R \cup A_1) = 1$ then A <- $A_1$
   else
      if $f(C \cup R \cup A_2) = 1$ then A <- $A_2$
      else {both $A_1$ and $A_2$ cover points covered by the critical
      vector}
         R <- $A_1 \cup R$
         A <- $A_2$
4. If A has more than two 1's, then return to step 2
   else
      C <- $A \cup C$
      A <- R
      R <- u.l.b
      if A = u.l.b or A = $x_p$ then C is a critical vector
      covering $x_p$
      else return to step 4.
```

Definition 7.4 There are monotone nets whose output does not depend on one of the inputs. If the output does not depend on input i, we say that the net is redundant with respect to input i.

Remark If the function f is redundant with respect to the input corresponding to x_p, the algorithm terminates, but it returns a vector that is not a critical one. Therefore, if the function may be redundant with respect to the considered input, it must be checked that the output of the algorithm is actually a critical vector.

Remark When 0 an 1 are the only allowable CF's and attenuations, probabilistic sum behaves like MAX. Therefore, the critical set algorithm can be used to find tests even for nets with the following parameter choices: (a) MIN and p+ combinators and integrators; (b) 0/1 weights (attenuators) and CF's; (c) identity predicate functions.

Synthesis of attenuations in MIN/MAX graphs with large CF alphabets

Consider the case of a finite CF alphabet different from {0,1}. The other parameters are: (a) MIN and MAX attenuators and combinators; (b) finite CF alphabet, {0,...,p}; (c) identity predicate functions.

It will be shown that the results obtained for 0/1 CF's and weights (attenuations) in the previous section extend to this more general case.

Problem name. Detection of incorrect attenuations, monotone input case in nets with large CF alphabets (DIML).

Problem instance. An inference net with parameters as described above. A function to be realized by the inference net. A marker for a special attenuator in the net. All weights in the net are one.

Question. Is there a test that detects whether the marked attenuator should have value less than one?

Comment The problem is equivalent to that of detecting a stuck-at-k fault on the input to a logic circuit using the M-logic, as defined, for example, in [Lu and Lee, 1984]. Therefore, one could use a method to detect fault in these circuits to solve problem T2. One such method is the M-Difference method, described in [Lu and Lee, 1984]. However, the method presented below is faster in the worst case than the M-difference method, for input tests in monotonic circuits.

Definition 7.5 A critical k-vector is a vector $\underline{x}$ such that:
(1) $f(\underline{x}) = k$;
(2) for all $\underline{x}_0 < \underline{x}$, $f(\underline{x}_0) <= k$.

Definition 7.6 A test k-vector for input i with value j is a k-vector $\underline{x}$ s.t.:
(1) $f(\underline{x}) = k$;
(2) $f(\underline{x}_i) <> k$, for all $\underline{x}_i$'s that differ from $\underline{x}$ in that the i-th element is less than in $\underline{x}$, i.e., it is less than j.

Definition 7.7 A vector $\underline{x}$ is a descendent of vector $\underline{y}$ if the components of $\underline{x}$ are pairwise less than or equal to the corresponding components of $\underline{y}$. We also say that $\underline{x}$ is covered by $\underline{y}$. If only one component of $\underline{x}$ is less than the corresponding component of $\underline{y}$ and by only one unit, we say that $\underline{x}$ is an immediate descendent of $\underline{y}$.

Example 7.3

For the three-input monotone function represented in Figure 7.3, vector 221 is a test 2-vector for input 1 s-a-2, because f(221) = 2, f(121) = 0; vector 211 is a test 2-vector for input 1 s-a-2, because f(211) = 2, f(111) = 0; vector 212 is a test 2-vector for input 1 s-a-2, because f(212) = 2, f(112) = 1; vector 222 is a test 2-vector for input 1 s-a-2, because f(222) = 2, f(122) = 1. Vector 211 is also a critical 2-vector, because all its descendants are <2-vectors. Vector 210 is a test 1-vector for input 1 s-a-2, because f(210) = 1, f(110) = 0. It is also a critical 1-vector, because all its descendants are 0-vectors.

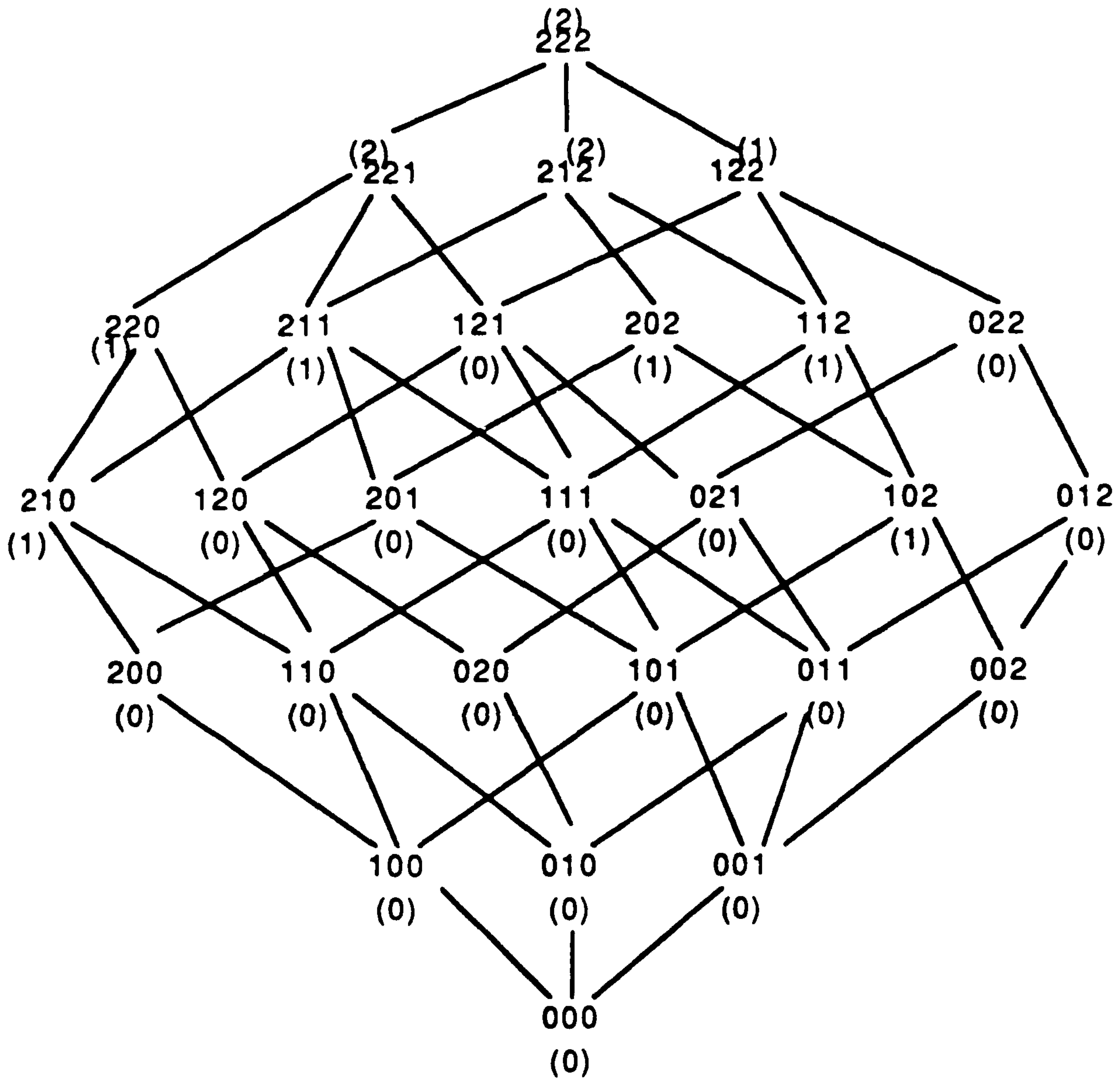


Figure 7.3 A three-input monotone function

(End of example)

<u>Test method</u> How to detect a single faulty input

attenuation. To find whether value j for input i should be attenuated to realize the desired function, find a test k-vector for input i with value j.

<u>Theorem 7.2</u> If there is a test k-vector for input i with value j, then there is a critical k-vector which covers the vector with value j for input i and 0's everywhere else.

Proof.

If there is no test, the theorem trivially holds.

Assume that there is a test.

Let $\underline{x}_p$ be the vector with 0's everywhere, but position i, where there is a j Any test $\underline{x}_t$ is such that $\underline{x}_t$ covers $\underline{x}_p$.

Say that $\underline{x}_l$ is a "lowest-level test" in the lattice of tests, i.e., the sum of its elements is smallest.

Now show that, if $\underline{x}_l$ is not a critical vector, then $\underline{x}_l$ is not a lowest-level test vector, establishing the desired result by contradiction.

Assume that $\underline{x}_l$ is not a critical vector. Give the name h to the value of input i in $\underline{x}_l$.

Since $\underline{x}_l$ is a test, its immediate descendent with j-1 as value of input i, $\underline{x}_{li/j-1}$, is a <k-vector, i.e. a vector $\underline{y}$ s.t. $f(\underline{y}) < k$.

Since $\underline{x}_l$ is not a critical vector, it must have an immediate (because of monotonicity) descendent which is a k-vector. Call this descendent $\underline{x}_{ld}$. $\underline{x}_{ld}$ must be a vector with h in the i-th position, because $\underline{x}_{li/j-1}$ is a <k-vector. Therefore, this descendent covers $\underline{x}_p$. $\underline{x}_{ld}$ is a test, because: (a) $\underline{x}_{ld}$ covers $\underline{x}_p$; (b) $\underline{x}_{ld}$ is a k-vector; (c) $\underline{x}_{ldi/j-1}$ is a <k-vector, since $\underline{x}_{ldi/j-1} < \underline{x}_{li/j-1}$ is a <k-vector, and f is monotonic.

Therefore, $\underline{x}_l$ is not a lowest-level vector which is a test, because it has a descendent, $\underline{x}_{ld}$, which is also a test. We have a contradiction.

Therefore, $\underline{x}_l$ is a critical vector.

(End of proof.)

<u>Algorithm 7.2</u> Find critical k-vectors.

(Assumption: there is a k-critical vector covering $\underline{x}_p$)

```
1. C <- $\underline{x}_p$
```

```
   A <- the greatest $x$ s.t. $x$ covers $x_p$ and f(A)=k
   R <- u.l.b.
2. Let $A_1$, $A_2$ be vectors s.t.
   (a) together, they cover all join-irreducible elements
   covered by A
   (b) each of them covers fewer join-irreducible elements than
   A
   (c) they all cover the same number of join-irreducible
   elements
3. If f(CuRu$A_1$) = k then A <- $A_1$
   else
      if f(CuRu$A_2$) = k then A <- $A_2$
      else
         R <- $A_1$uR
         A <- $A_2$
4. If A is not a join-irreducible vector or the u.l.b., then
   return to step 2 else
      if there is an E s.t. A covers E and f(CuRuE) = k, then
      let A be the smallest such E
      C <- AuC
      A <- R
      R <- u.l.b
5. If A = u.l.b then C is a critical vector covering $x_p$
   else return to step 4.
```

<u>Proof of termination</u>

We identify all loops and associate a natural number to the first statement of each loop. It must be shown that this number decreases for each execution of the loop. Instead of a formal proof, I will argue informally.

There are only two loops in the program. The <u>large loop</u> involves steps 2 and 4. The <u>tight loop</u> involves step 4 and the last line of the program. In both cases, we assign to the first line of the loop the level of vector A in the lattice of vectors.

Because of condition (b) in step 2, A_1 and A_2 have lower level than A. Therefore, R just before step 4 has lower level than A. Therefore, A at the exit point has always lower than at step 2.

For what concerns the tight loop, note that at the last line, the following holds: A <> u.l.b, R = u.l.b. If A is a join-irreducible vector, the condition at step 4 holds and the program terminates; if A is not a join-irreducible vector, we return to step 2 and are therefore of the tight loop, with an A of a lower level than when we entered the tight loop.

(End of proof of termination.)

Proof of partial correctness

It is assumed that there exists a critical k-vector covering x_0 and that f(A) = k. It will be shown that, when exiting at step 5, C is a critical vector covering x_0, i.e., (1) f(C) = k and (2) $f(C_0) < k$, where C_0 is strictly covered by C.

(1) holds because it is always true that f(CuRuA) = k: it is true after initialization (i.e., just before step 2) and it is invariant for both the large and the tight loops. But at exit, A = u.l.b and R = u.l.b, therefore f(CuRuA) = k implies f(C) = k.

Now, it will be shown that (2) holds.

Consider C as built of the join of join-irreducible vectors. Each of these vectors is obtained by "splitting" A or by descending along a chain of join-irreducible vectors in the step after 4. An example of a chain of join-irreducible vectors: 030-020-010.

Steps 3-4 guarantee that each of the join-irreducible vectors must be present in order for f(C) = k to hold; the step after 4 insures that each join-irreducible vector is the least possible. In other words, steps 3-4 isolate a chain of join-irreducible vectors that must be present in C for f(C) to be k; the step after 4 isolates the least vector from that chain that still lets f(C) be k.

(End of proof of partial correctness.)

Note that property (c) in step 2 is neither used to prove termination nor to prove partial correctness.

Complexity

Complexity depends on how efficient it is to "split" A into A_1 and A_2 in step 2. By insuring that property (c) always holds at step (2), the space of vectors covered by A is always (approximately) halved each time step 2 is executed. This is insured by splitting a vector by substituting half of its elements with 0 and leaving the others unchanged. Sometimes, it may be more efficient to split a vector in some other way, but other strategies cause A_1 and A_2 not to be disjoint, which means that the number of join-irreducible vectors is not halved each time step 2 is performed.

Here are some examples of vector splits. The vector space is halved in the first two cases, while it is not halved in the third example. 6666 -> 6600, 0066. 6534 -> 6500, 0034. 6666 -> 6655, 5566.

Call the vector size n and the cardinality of the CF alphabet m. Using the 6666 -> 6600, 0066 strategy, one must deal with only n chains of join-irreducible vectors at step 2. Then, a vector (vector "E") in each chain must be searched in the step after 4. In all, O(m*logn) calls to f are necessary to individuate <u>one</u> element of vector C. Since there are n elements in C, O(n*m*logn) = O(mnlogn) calls to f is the total complexity.

The algorithm can be modified to run in the case in which A is initialized to a vector **x** such that f(**x**) > k. Here is the modified algorithm.

<u>Algorithm 7.3</u>

```
   {Assumption: f(u.u.b) >= k}

1. C <- $x_p$
   A <- u.u.b.
   R <- u.l.b.
2. Let $A_1$, $A_2$ be vectors s.t.
   (a) together, they cover all join-irreducible elements
   covered by A
   (b) each of them covers fewer join-irreducible elements than
   A
   (c) they all cover the same number of join-irreducible
   elements.
3. If f(CuRu$A_1$) >= k then A <- $A_1$
   else
      if f(CuRu$A_2$) >= k then A <- $A_2$
      else
         R <- $A_1$uR
         A <- $A_2$
4. If A is not a join-irreducible vector or the u.l.b. then
   return to step 2 else
4.1   let E be the smallest E covered by A and s.t. f(CuRuE)=k
      C <- AuC
      A <- R
      R <- u.l.b
5.    if A = u.l.b then C is a critical vector covering $x_p$
      else
         return to step 4.
```

The proof of termination is totally analogous to that for the previous algorithm. In the proof of partial correctness, a different argument must be used to show that f(C) = k at step 5 at termination. Now, what is invariant for both the large and the tight loops is f(CuRuA) >= k. But at exit, because of step 4.1, f(C) = k.

CHAPTER EIGHT

APPROXIMATIONS AND INTERMEDIATE VALUES

Introduction

The previous four chapters have shown that the problem of synthesizing attenuations is hard, at least for moderately complex topologies in the incomplete case. It is therefore natural to investigate how difficult it is to obtain **approximate** solutions to this hard problem and whether more comprehensive tests that include the certainty factors for **intermediate hypothesis** simplify the synthesis problem. (Definitions will be given in the appropriate sections of this chapter.)

Approximations are considered in section one. The use of tests that include the certainty factors of intermediate hypotheses is considered in section two.

Approximations

Consider the problem of synthesizing approximate attenuations from tests (complete case). One could expect to be able to find an efficient algorithm to synthesize approximate attenuations within a certain error of the correct ones. It will be shown that this is not possible, at least for a suitable, but reasonable, definition of approximation.

Problem name. Approximate Restricted Attenuation Synthesis (ARA).

Problem Instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real values between 0 and 1 at the output of MIN boxes, bounded fan-in to MIN's, bounded depth; a set of tests.

Question. Is there an assignment of attenuations that is within less than 0.5 from the correct assignment?

Theorem 8.1. ARA is NP-Hard.

Proof.

If it were possible to solve ARA in polynomial time, it would be possible to solve problem RA in polynomial time (therefore contradicting Theorem 5.1), by using the following algorithm: firstly, compute approximate attenuations using ARA; secondly, set each attenuation that is less than .5 to 0 and each each attenuation that is more than .5 to 1. (Note that no attenuation will be equal to .5, because ARA obtains attenuations which are within less than, not less than or equal to .5 of the correct assignment.) Clearly, this algorithm is correct and runs in polynomial time if ARA does.

(End of proof.)

The same proof technique can be used to show that attenuation synthesis is NP-Hard when probabilistic sums are used in place of MAX's. The details are left to the reader.

Intermediate Values

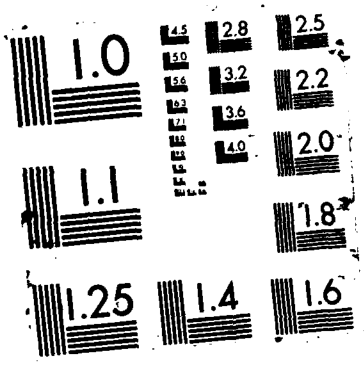
In this section, we assume that information about the strengths at nodes internal to the inference net is available. This differs from the rest of the thesis, where we only assume that information is available about the input-output relation, in the form of tests (incomplete case) or perfect experts (complete case). Clearly, this new model allows for more information to be available to synthesize (or refine) tests, and one would expect that efficient synthesis procedures would exist.

Two cases are considered: in the first one, intermediate certainty factors are attached to the output of attenuators; in the second one they are attached to the output of combinators and integrators. Figures 8.1 and 8.2 illustrate the first and the second case, respectively, on simple examples. In both cases it is possible to determine attenuations independently of each other, on a local basis. This trivializes the synthesis problem.

AD-A185 626 AUTOMATING RULE STRENGTHS IN EXPERT SYSTEMS(U) DUKE UNIV DURHAM NC DEPT OF COMPUTER SCIENCE M G VALTORTA MAY 87 AFOSR-TR-87-1348 $AFOSR-83-0205 2/2

UNCLASSIFIED F/G 12/9 NL

END


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

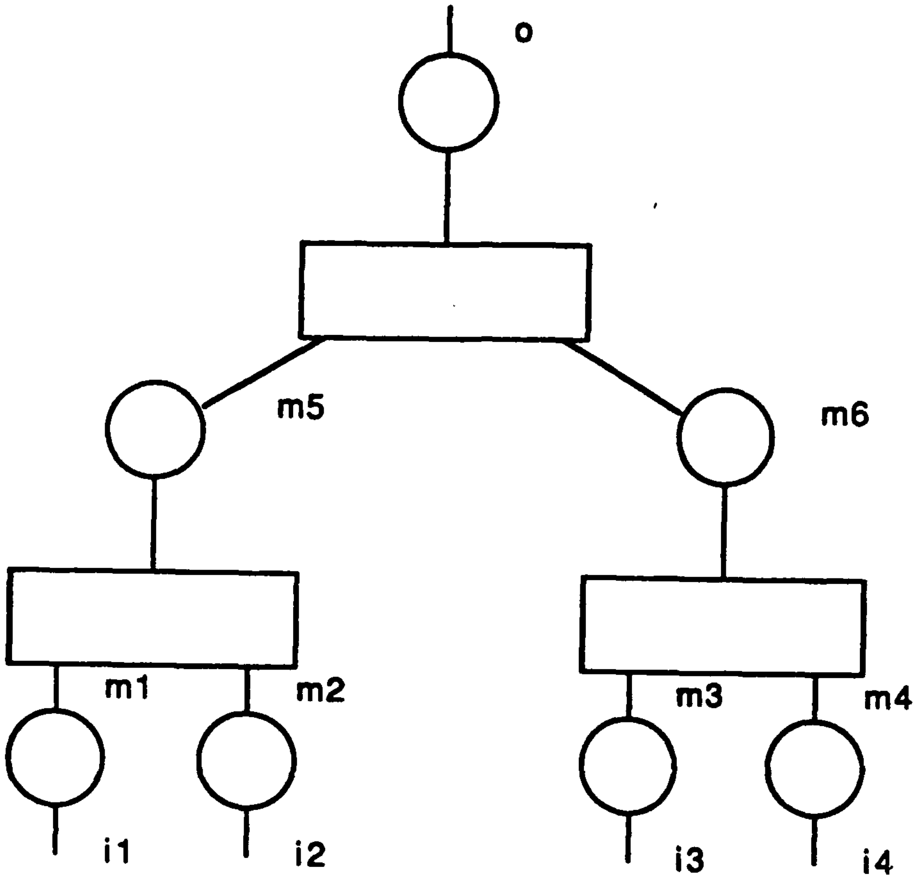


Figure 8.1 If the m_i's are given, attenuations can be synthesized easily

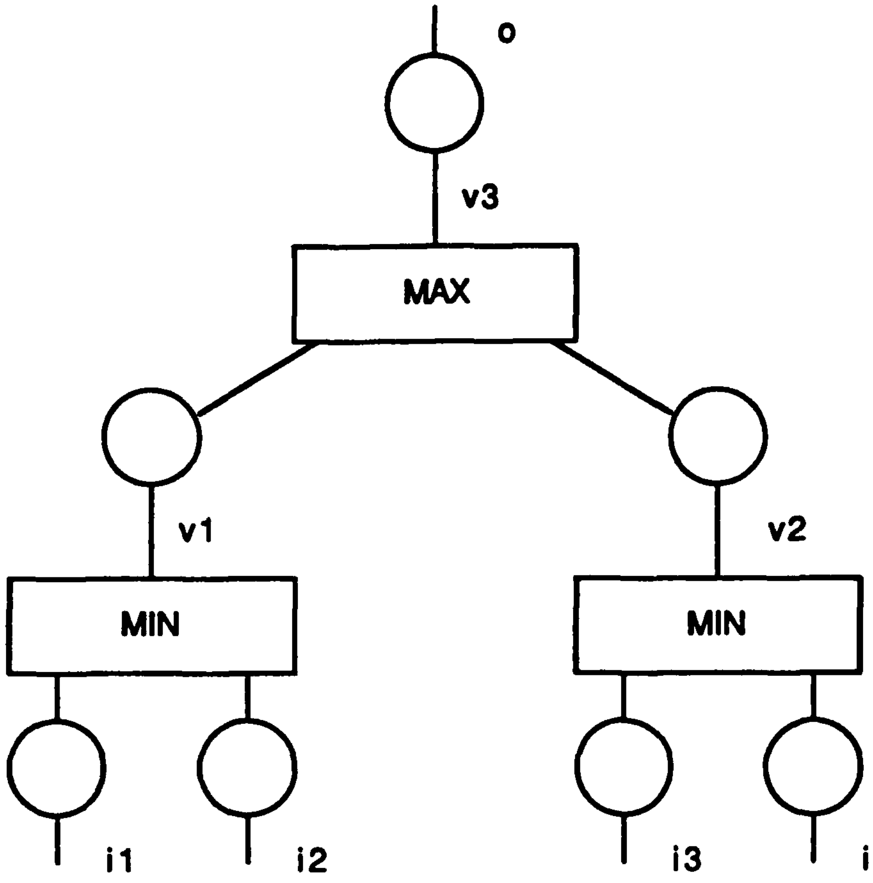


Figure 8.2 If the v_i's are given, it is easy to synthesize attenuations

It is evident how attenuations can be synthesized when the m_i's are given. It will now be shown how to synthesize attenuations when the v_i's are given. Providing the v_i's divides the problem of synthesizing the attenuation for the whole net into that of attenuation synthesis for nets containing only one MIN or one MAX box. The MIN and MAX cases will be presented separately. Consider the MAX case first. Assume that a MAX box has q inputs. (Refer to Figure 8.3 for notation.)

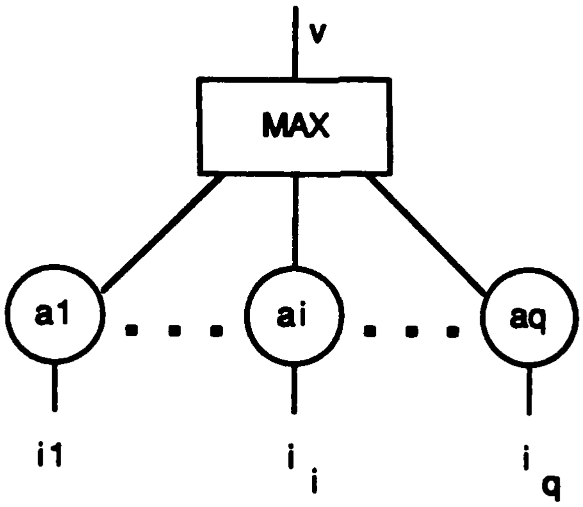


Figure 8.3 MAX subgraph for the v_i case

Tests are given. The generic test is $T_j=((i^j_1,\ldots,i^j_i,\ldots,i^j_q),v^j)$. Define $a^j_i=v^j/i^j_i$. The generic attenuation is a_i=min a^j_i. We claim that the tests are satisfied only if the attenuations are set as just stated. In fact, if attenuation a_i is set to a larger value, there will be a test, T_k, for which the output of the box is large than v^k, and if a test is not satisfied with these values for the attenuations, it will certainly not be satisfied with smaller attenuations.

The MIN case is totally analogous. Notation is as for the MAX case. The generic attenuation is a_i=max a^j_i. The tests are satisfied only if the attenuations are set as just stated.

Gallant [1985] studied a similar model in which all intermediate values are given, but the only integrator and combinator function is the sign of a sum. In Gallant's model, the certainty factor of each variable is determined by evaluating a linear combination of the certainty factors of the variables that appear in antecedents of rules concluding about the variable in question. Integrators and combinators are algebraic sums. However, the value obtained by summing is then mapped into -1, +1, and 0, depending on whether it is negative, positive, or zero. Attenuations (which can be positive, zero, or negative) are used as weights in the sums of certainty factors that determine the values of certainty factors for all variables that occur in the conclusion of some rule. For example, in the net illustrated in Figure 8.4, certainty factor c is the sign of $i_1a_1+i_2a_2+i_3a_3$ and certainty factor o is the sign of ca_7+da_7.

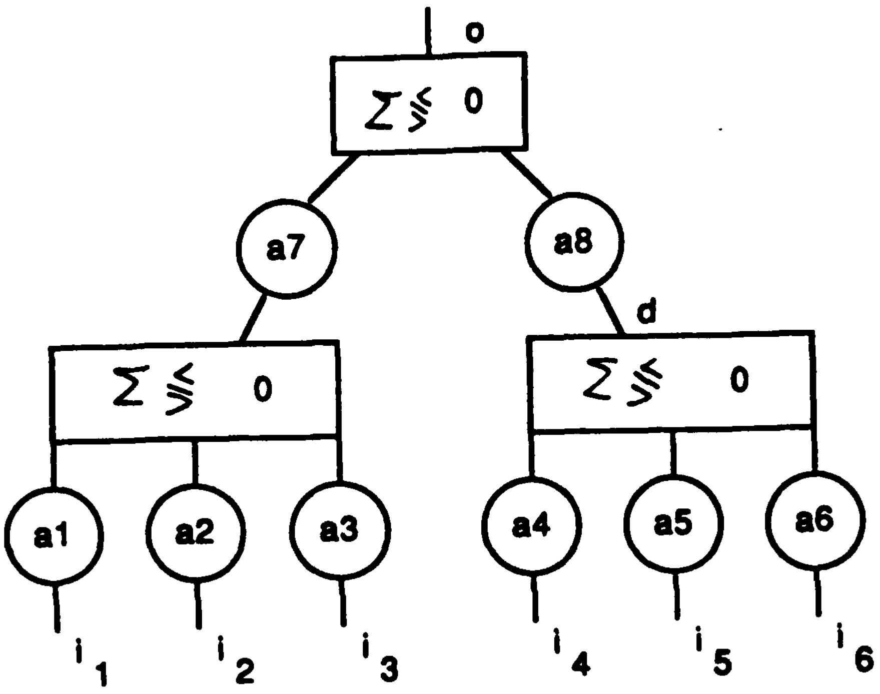


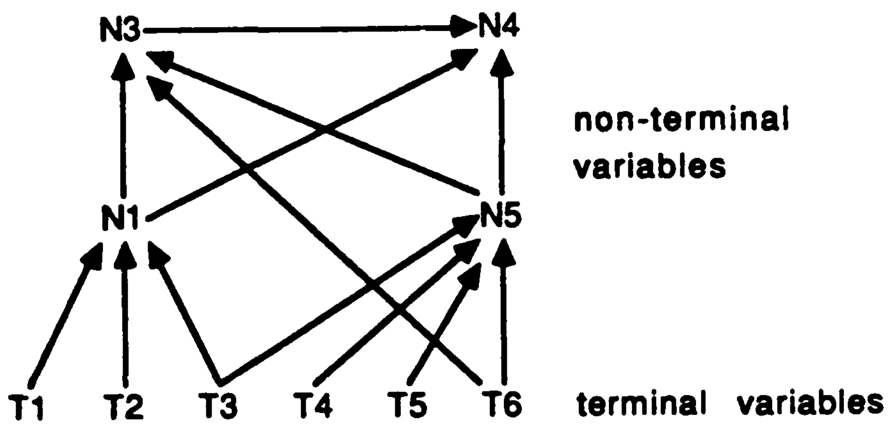
Figure 8.4 Sample inference network using Gallant's model

We now describe the model in more detail using Gallant's terminology and explain a method for computing attenuations from cases. The reader is invited to refer to [Gallant, 1985] for details. Inference nets are acyclic graphs; CF values are -1, +1, and 0, to be interpreted as no, yes, and unknown, respectively.

Rules are propositional and negation is built into the model. Examples of rules are

1. If N_1 AND NOT T_6 Then N_3.
2. If N_2 AND NOT T_6 Then N_3.

Variables are partitioned in <u>terminal variables</u> and <u>non-terminal variables</u>. Partitioning is done by statically analyzing the rule base: terminal variables do not occur in conclusions of rules; non-terminal variables occur in the conclusion of some rules. This partition can be represented graphically and in tabular form as in the example in Figure 8.5. The table is called dependency matrix. The rows of the dependency matrix are labeled by the non-terminal variables; its columns are labeled by all the variables. A 1 in position (i,j) means that there is a rule with variable i in its conclusion and variable j in its premise. A 0 in position (i,j) means that there is no such rule. Cases are assignments of true, false, and unknown to all variables. The last two lines in Figure 8.5 represent two cases.



	T_1	T_2	T_3	T_4	T_5	T_6	N_1	N_2	N_3	N_4
N_1	1	1	1	0	0	0	0	0	0	0
N_2	0	0	1	1	1	1	0	0	0	0
N_3	0	0	0	0	0	1	1	1	0	0
N_4	0	0	0	0	0	0	1	1	1	0

Cases:

T	T	T	F	F	F	T	F	T	F
F	F	F	T	T	F	F	T	T	F

Figure 8.5 An acyclic inference net, a dependency matrix, and two cases

The learning matrix has as many rows as the dependency matrix. Each row in the learning matrix is the linear discriminant for the corresponding non-terminal variable. This means that, interpreting -1 as false, 0 as unknown and 1 as true, row i holding vector L_i determines the value for its intermediate or goal variable N_i according to the rule:

$$N_i = \begin{cases} +1 \\ -1 \\ 0 \end{cases} \text{if } L_i.V = \sum_{j=0}^{m} L_{ij}*V_j \begin{cases} >0 \\ <0 \\ =0 \end{cases}$$

(where $V_0 = 1$).

Figure 8.6 gives an example of learning matrix, for the net whose dependency matrix was given in Figure 8.5. For example, if T_1, T_2, and T_3 are true, as in the first case in Figure 8.5, then N_1 would also be true, because its linear discriminant evaluates to a value greater than zero, namely -1+3-3+3, i.e., 2.

$$\begin{array}{cc|cccccccccc}
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 & & C & T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & N_1 & N_2 & N_3 \\
1 & N_1 & -1 & 3 & -3 & 3 & & & & & & \\
2 & N_2 & 1 & & & 3 & 3 & 3 & & & & \\
3 & N_3 & -2 & & & & & & -4 & 2 & 2 & \\
4 & N_4 & -2 & & & & & & & 2 & 2 & -4
\end{array}$$

Figure 8.6 An example of a learning matrix.

The learning matrix is inferred from the cases by using any of a variety of techniques: Gallant argues for the use of an iterative method, called the Pocket Algorithm, but linear programming can also be used. For example, the two cases in Figure 8.5 are compatible with the dependency matrix in Figure 8.5 if and only if the following linear system has a solution:

$L_{01} + L_{11} + L_{21} + L_{31} > 0$ (first case: N_1 is T when T_1, T_2, T_3 are T)

$L_{02} + L_{32} - L_{42} - L_{52} - L_{62} < 0$ (first case: N_2 is F when T_3 is T and T_4, T_5, T_6 are F)

$L_{03} - L_{63} + L_{73} - L_{83} > 0$ (first case: N_3 is T when T_6 and N_2 are F, N_1 is T)

$L_{04} + L_{74} - L_{84} + L_{94} < 0$ (first case: N_4 is F when N_1, N_3 are T, N_2 is F)

$L_{01} - L_{11} - L_{21} - L_{31} < 0$ (second case: N_1 is F when T_1, T_2, T_3 are F)

$L_{02} - L_{32} + L_{42} + L_{52} - L_{62} > 0$ (second case: N_2 is T when T_3 and T_6 are F, T_4 and T_5 are F)

$L_{03} - L_{63} - L_{73} + L_{83} > 0$ (second case: N_3 is T when T_6, N_2 are F, N_1 is T)

$L_{04} - L_{74} + L_{84} + L_{94} < 0$ (second case: N_4 is F when N_1 is F, N_2 and N_3 are T).

CHAPTER NINE

REFINEMENT

Introduction

The problem of synthesizing approximate attenuations has been shown to be NP-Hard in the previous section, for a simple definition of approximation and even for the restricted network topology introduced in chapter 5. (The previous chapter has also contained a discussion of the synthesis of attenuations when the certainty factors of intermediate hypothesis are known.) The results of the previous chapter make it necessary to investigate the use of expert-given attenuations as a starting point for the refinement of rule strengths.

The first section considers whether knowing principal paths (defined in section 3 of chapter 4) for all tests helps synthesizing attenuations in the incomplete case. The second section contains two NP-Hardness results concerning the problem of refining attenuations starting from very good estimates. The third section exploits the structure of the proofs of the results presented in section 2 to speculate as to why refinement is a hard problem and, more concretely, present a fast algorithm for refinement when a simple condition on the expert-given attenuations holds. The fourth section presents several iterative algorithms for attenuation refinement, based on Rada's work [1984; 1985] and considers their convergence properties using the results obtained in the previous sections; this analysis is used to suggest methods for the refinement of attenuations under more general conditions than those described in the previous section.

Synthesis of attenuations when principal paths are known

In this section, all non-identity attenuations are at the leaves, without loss of generality. (See section 2, "independent attenuations," in chapter 4.)

It is easy to synthesize attenuations in a MIN/MAX tree with n inputs, if n tests with different, known principal paths are given.

For example:

(a) in the case for which attenuations are restricted to be 0 or 1, n tests are given that have input equal to output for only one input, which is different for each test;

(b) in the "real numbers" case, each test has only one input greater than or equal to the output and this input is different for each test.

Note that it is <u>not</u> necessarily easy to synthesize attenuations if we only have n tests whose i/o ratios are all different, because that just means that no two tests can share the same influential path but does not give any information, in general, as to which path is associated with each test.

One could try to refine a rule base without changing influential paths, or, if test cases are misclassified using the expert-given attenuations, one could first try to modify attenuations in such a way that:

(a) the test cases are correctly classified;

(b) principal paths are the same as with the expert-given attenuations.

If no satisfactory refinement were possible, while keeping influential paths unchanged, one could try changing the assignment of principal paths and solving a similar problem with a new assignment of principal paths. For such an approach to succeed, it is necessary that the following problem be solvable quickly.

Problem name. Restricted Attenuation Synthesis with Known Principal Paths (RASP).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative 0/1 attenuations, bounded fan-in to MIN nodes, bounded depth; a set of tests; a principal path for each test (i.e., a function f from tests to paths).

<u>Remark</u> The set of RASP problem instances is not a subset of the set of RA problem instances, defined in chapter 5. However, each RASP instance can be trivially mapped in polynomial time into a corresponding RA instance by removing the "principal path for each test" from the RASP instance. Therefore, the NP-Hardness of RASP implies the NP-Hardness of RA. Since the proof of NP-Completeness of RASP given below draws heavily from the simpler proof that RA is NP-

Complete, given in chapter 5, both proofs are given for ease of exposition.

Question. Is there an assignment of attenuations for which all tests are handled correctly?

Theorem 9.1 RASP is NP-Complete.

Proof

RASP is in NP, because it can be solved in polynomial time by a nondeterministic algorithm that loops through all possible assignments of 0 and 1 to the attenuations. It must now be shown that RASP is NP-Hard. For this purpose, it will be demonstrated that MSAT (Monotone 3-Conjunctive Normal Form Satisfiability) is reducible to RASP, by providing an algorithm that builds, in polynomial time, an instance of RASP given an expression E in MSAT, such that the RASP instance is a yes-instance if and only if E is satisfiable. (MSAT is defined in section 2 of chapter 5.)

Let E be a MSAT expression with m clauses and n distinct variables. Build a tree as shown in Figure 9.1 below.

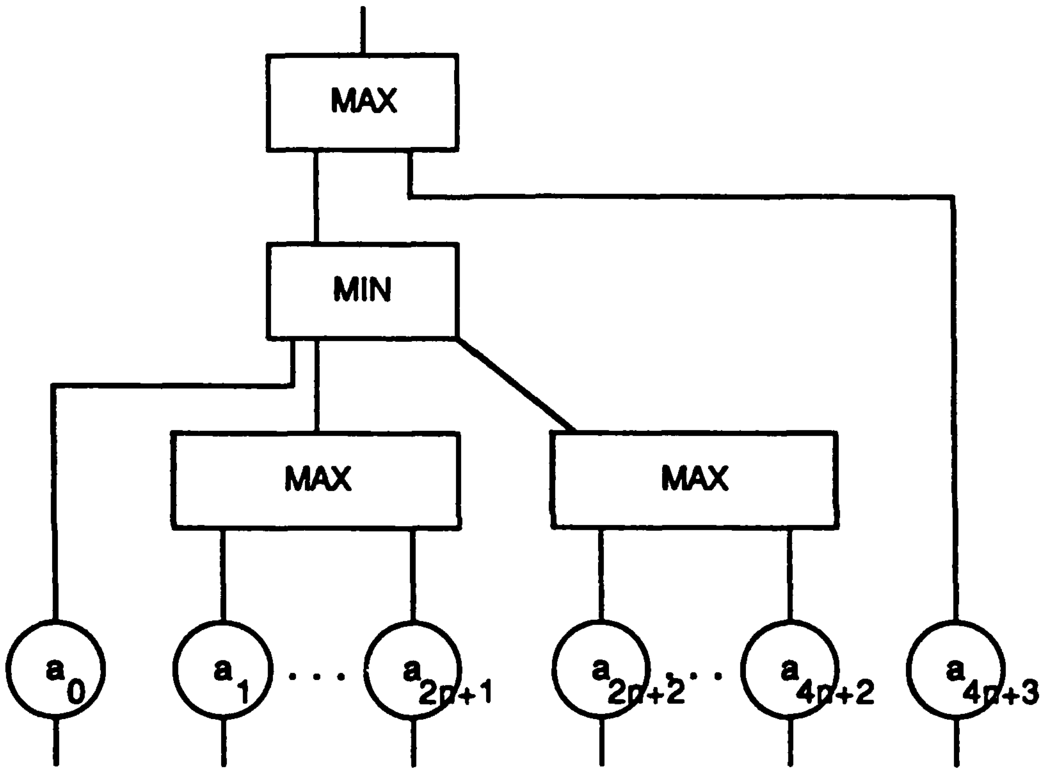


Figure 9.1 Generic RASP instance

There are 3+3n+2m tests for the RASP instance, built as follows. The first test has .5 in position 0, .9 in positions 1 through 4n+2, 0 in position 4n+3 and output .5. The second test has .6 in position 0, .9 in positions 1 through 4n+2, 0 in positions 4n+3 and output .6. The third test has 0 in positions 0 through 4n+2 and .2 in position 4n+3 and output .2. (These three tests insure that any solutions must have $a_0=a_{4n+3}=1$.)

m tests are built as the clause tests in the proof that RA is NP-Complete (in section 2 of chapter 5), except that each test is extended to the left with .5 and to the right with .4; the output for each of these tests is .5. m tests are built as the clause tests in the proof that RA is NP-Complete, except that each test is extended to the left with 1 and to the right with .8; the output of each test is .8.

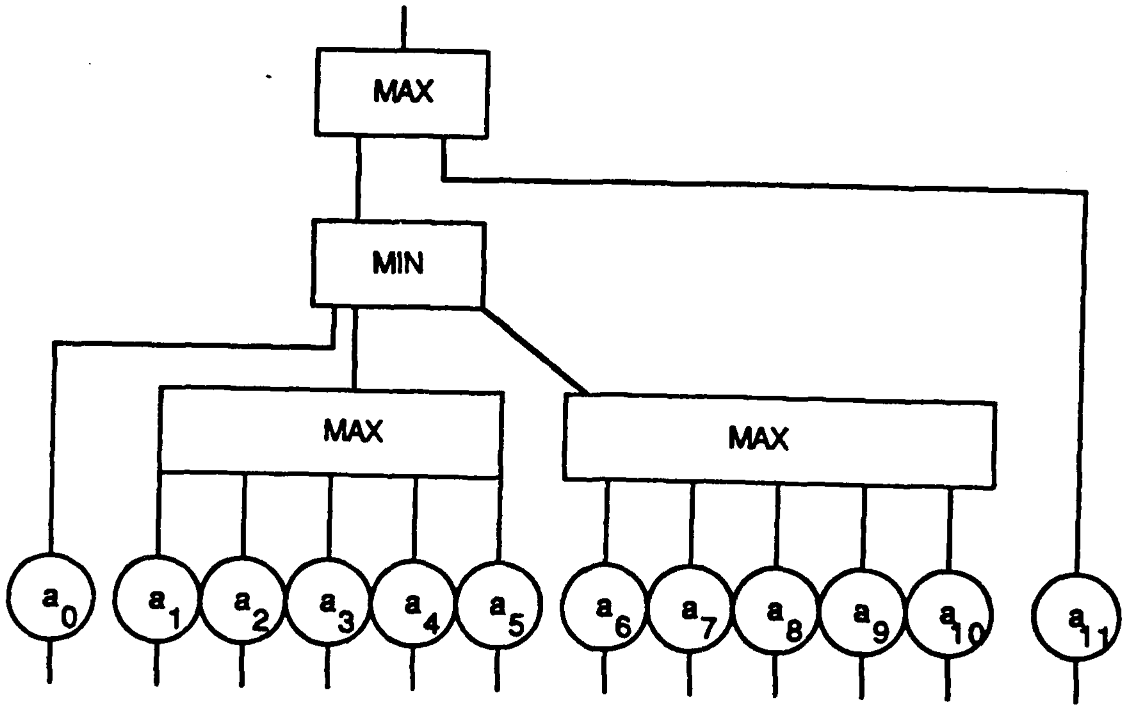
n tests are built as the first n variable tests in the proof that RA is NP-Complete, except that each test is extended to the left with 1 and to the right with .2; the output of each test is .2. n tests are built as the last n variable tests in the proof that RA is NP-Complete, except that each test is extended to the left with 1 and to the right with .7; the output of each test is .7. Finally, n tests are built as the preceding n, except that each test is extended to the left with .5, to the right with .4, and the output of each test is .5.

To complete the description of the RASP instance, the principal paths for the tests will be specified as follows: the principal path (PP) for the first and second tests includes a_0; the PP for the third test includes a_{4n+3}; the PP for the next m tests includes a_0; the PP for the next m tests includes a_{4n+3}; the PP for the last 3n tests includes a_{4n+3}.

An example of how MSAT instances are mapped into RASP instances is given below. Each clause in the example contains only two literals, for ease of presentation.

Example 9.1

The MSAT instance is $E=(x_1 v x_2)\&(-x_1 v x_2)$. (Therefore, m=2, n=2.) The corresponding RASP instance is shown in Figure 9.2. T_1 and T_2 force a_0 to be 1; T_3 forces a_1 to be 1; T_4 and T_5 are the clause tests corresponding to $(x_1 v x_2)$; T_6 and T_7 are the clause tests corresponding to $(-x_1 v -x_2)$; T_8, T_9, and T_{10} are the variable tests corresponding to x_1; T_{11}, T_{12}, and T_{13} are the variable tests corresponding to x_2.



													output
T_1	.5	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	0	.5
T_2	.6	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	0	.6
T_3	0	0	0	0	0	0	0	0	0	0	0	.2	.2
T_4	.5	.6	.6	0	0	0	0	0	0	0	.9	.4	.5
T_5	1	.6	.6	0	0	0	0	0	0	0	.9	.8	.8
T_6	.5	0	0	0	0	.9	.6	.6	0	0	0	.4	.5
T_7	1	0	0	0	0	.9	.6	.6	0	0	0	.8	.8
T_8	1	.6	0	0	0	0	.6	0	0	0	0	.2	.2
T_9	1	.6	0	.9	0	0	.6	0	.9	0	0	.7	.7
T_{10}	.5	.6	0	.9	0	0	.6	0	.9	0	0	.4	.5
T_{11}	1	0	.6	0	0	0	0	.6	0	0	0	.2	.2
T_{12}	1	0	.6	0	.9	0	0	.6	.9	0	0	.7	.7
T_{13}	.5	0	.6	0	.9	0	0	.6	.9	0	0	.4	.5

Figure 9.2 RASP instance corresponding to $(x_1 v x_2) \& (-x_1 v -x_2)$

(End of example)

To simplify presentation, redraw the network in Figure 9.1 as in Figure 9.3:

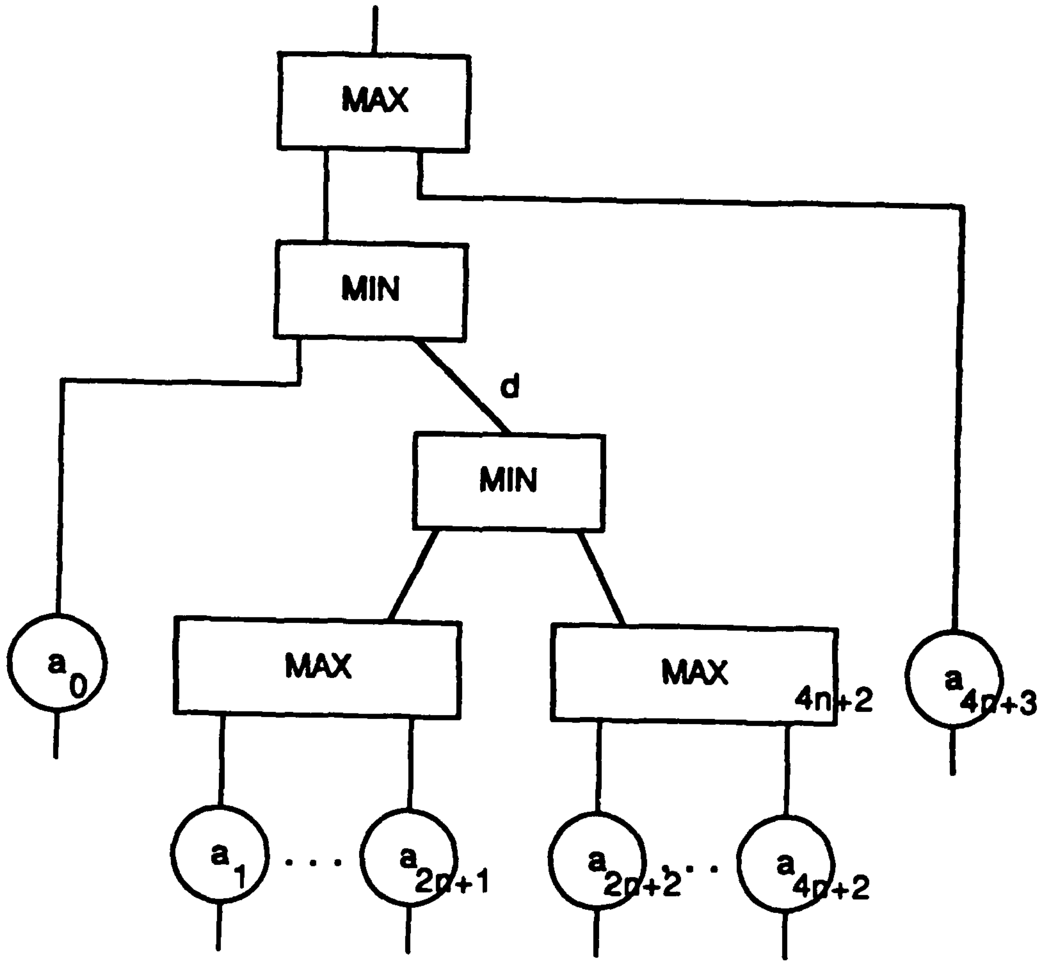


Figure 9.3 Generic RASP instance, redrawn

Consider the tree rooted at d. The purpose of the first three tests has been explained when they have been defined. The following 2m tests are built in such a way that in each yes-instance of RASP, the value of d for each test is .6, since the first m tests insure that that the value is greater than .5, the second m tests insure that the value is less than .8 and the only possible values are 0, .6, and .9. Similarly, the next n tests are built in such a way that the value on d is 0 and the last 2n tests are built in such way that the value on d is .6.

Therefore, the RASP instance is a yes-instance if and only if the RA instance, built as in the NP-Completeness proof of RA, is a yes-instance. But this RA instance is a yes-instance if and only if E is satisfiable. Therefore, the RASP instance, for whose construction a polynomial-time algorithm has been given, is a yes-instance if and only if E is satisfiable. This completes the proof that RASP is NP-Complete.

(End of Theorem)

Problem name. Restricted Attenuation Synthesis with Real CF's and Attenuations with Known Principal Paths (RAPRP).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real weights at the output of MIN boxes, bounded fan-in to MIN nodes; bounded depth; a set of tests; a principal path for each test (i.e., a function e from tests to paths).

Theorem 9.2 RAPRP is NP-Hard.

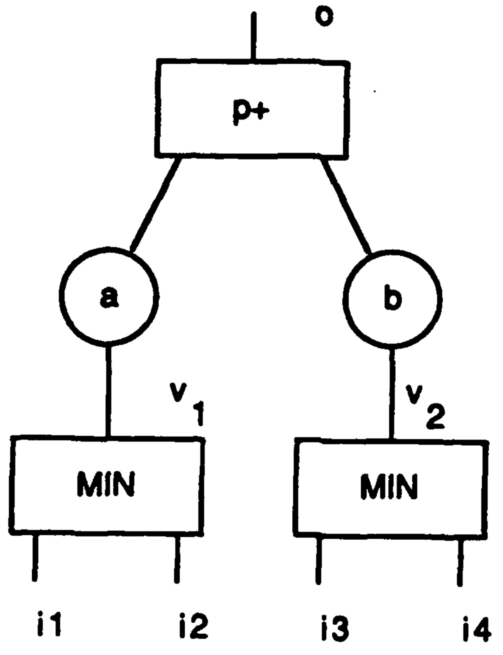
Proof.

The proof is an adaptation of the proof that AS is NP-Hard, in section 2 of chapter 5, modified as in the proof that RASP is NP-Complete. Details are left to the reader.

(End of proof)

In order to extend the problem corresponding to RASP (and RAPRP) to the MIN/p+ case, it is convenient to introduce a definition, which we first motivate with the following example.

Example 9.2



I_1 = (.6,.8,.5,.4)

Figure 9.4 Sample MIN/p+ inference tree

Consider the input part and the MIN/p+ tree in Figure 9.4. The following certainty factors flow in the tree: v_1 = .6, v_2 = .4, o = .6a[p+].4b. Clearly, there is no influential path in this example, since o is not equal to .6a or .8a or .5b or .4b. However, the output of the tree is equal to the output of the tree in Figure 9.5, which has no MIN boxes.

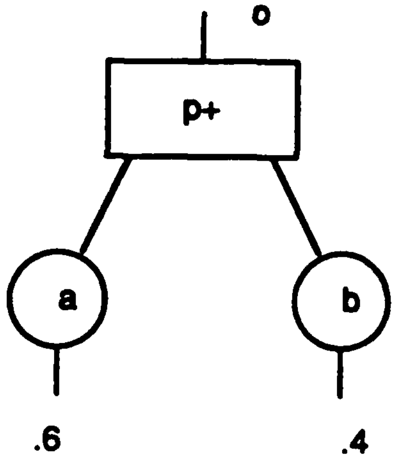


Figure 9.5 Influential bundle for the tree in Figure 9.4

(End of example)

In general, given an input part and a MIN/p+ tree, it is possible to define the MIN/p+ tree, without MIN boxes, which has the same output of the given MIN/p+ tree for the given input part.

Definition 9.1 Given a MIN/p+ tree and the input part of a test, an influential tree is a tree obtained by (1) removing each MIN boxes, starting from the ones closest to the leaves, and substituting the MIN box with the line on which flows the smallest CF value; (2) removing from the tree obtained in this way all subtrees whose root carries a CF of value 0. An influential tree is also called an influential bundle, a principal tree or a principal bundle.

Note that there may be several influential trees for a given tree and input part, depending on the tie-breaking rule used, as in the following example.

Example 9.3

The tree and input part shown in Figure 9.6(a) have the influential bundles indicated in Figure 9.6(b) and Figure 9.6(c).

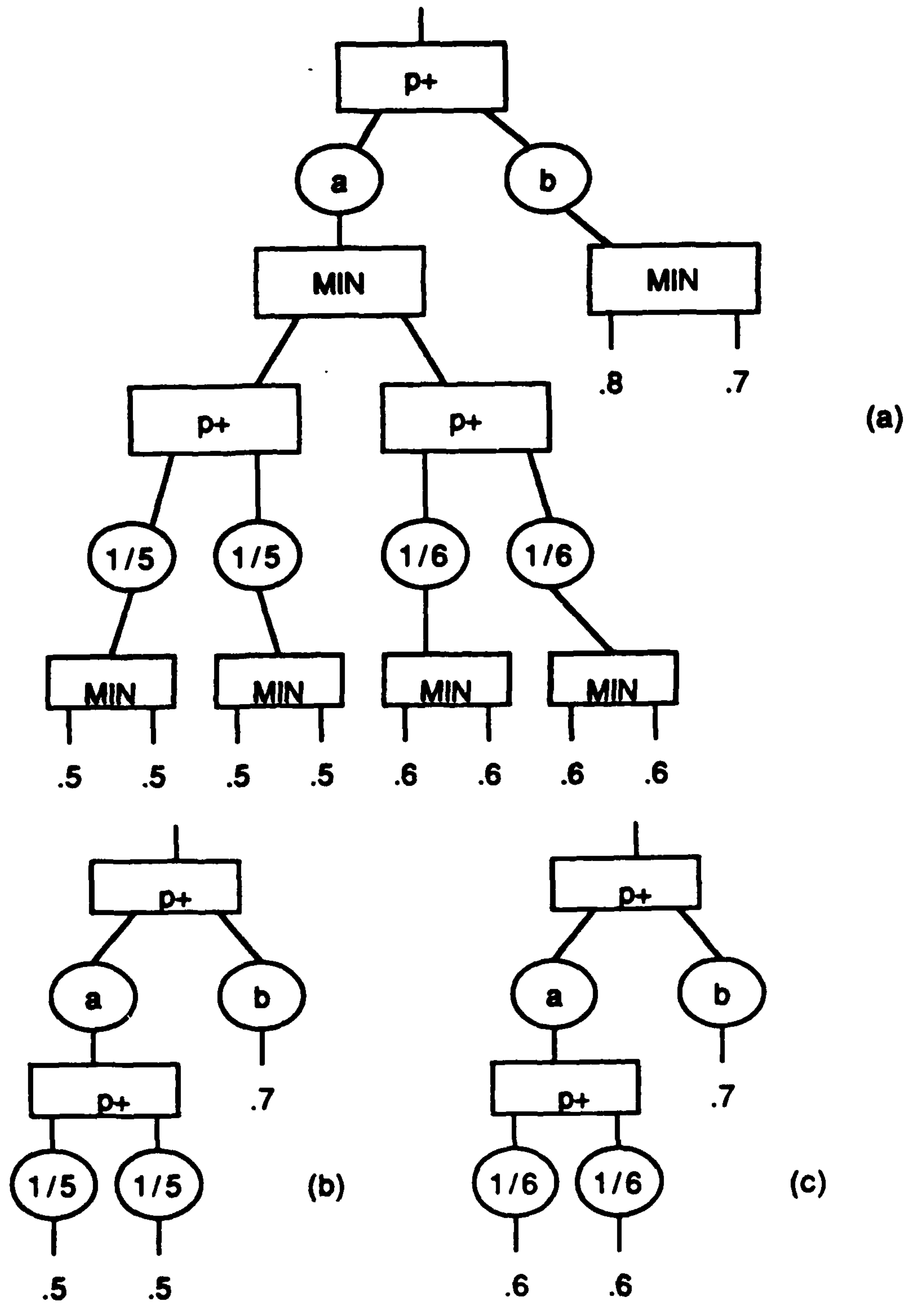


Figure 9.6 Inference tree and its influential bundles

(End of example)

It is also possible that an influential tree collapse into an influential path, as in the following example. In fact, tests that obtain this result were used in the algorithm described in section 5 of chapter 4 ("synthesis of attenuations: MIN/probabilistic sum case").

Example 9.4

The tree and test shown in Figure 9.7(a) have the influential bundle shown in Figure 9.7(b).

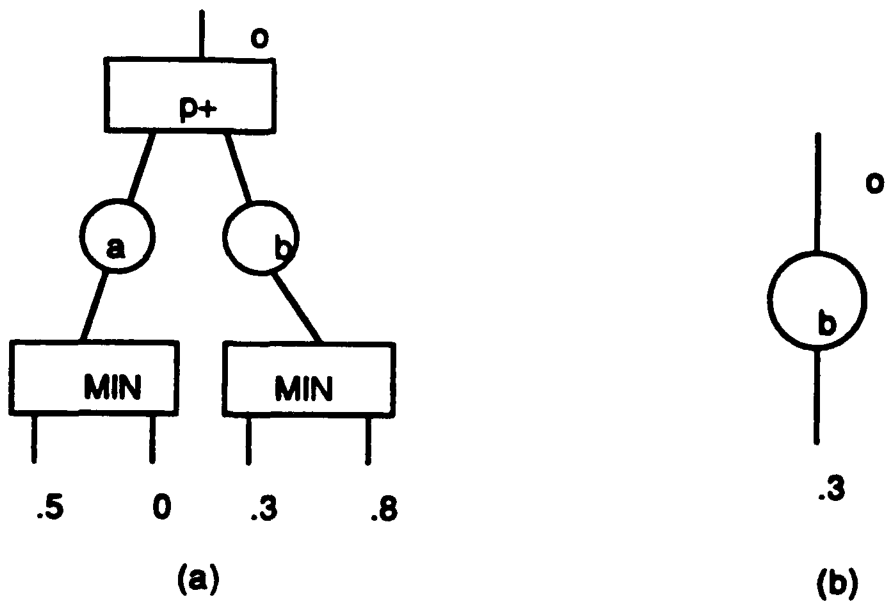


Figure 9.7 Inference tree and its influential path

(End of example)

One can only consider tests for which influential trees collapse into influential paths to show that the synthesis problem with known principal bundles is NP-Hard, since influential paths are a special case of influential bundles. The problem is formalized below.

Problem name. Restricted Attenuation Synthesis with Known Principal Paths, MIN/p+ case (RASPp+).

Problem instance. A tree with alternating MIN and p+ boxes, multiplicative attenuators with real weights at the output of MIN boxes, bounded fan-in to MIN nodes, bounded depth; a set of tests; a principal path for each test (i.e., a function f from tests to paths).

Theorem 9.3 RASPp+ is NP-Complete.

Proof.

This proof is only sketched, since it is similar to the preceding two.

RASPp+ is in NP.

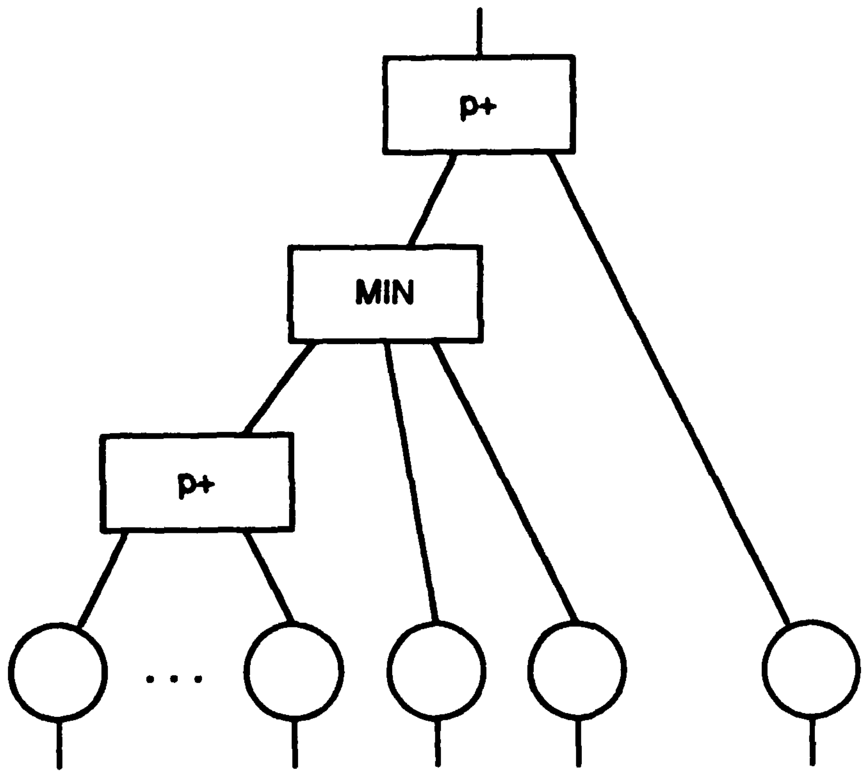


Figure 9.8 Generic RASPp+ instance

RASPp+ is NP-Hard. This proof follows the same line as the proof that RASP is NP-Hard: we build a network and a set of tests such that RASPp+ not being NP-Hard would imply that RAP is not NP-Hard. (The proof that FAP is NP-Hard is given in section 3 of chapter 5.) The network is the tree shown in Figure 9.8. The input parts of the tests in the RASPp+ instance, as the reader may have guessed by now, are made of the juxtaposition of the input parts of the tests in the RAP proof, the output of those tests, and 0. The output part of each test is the same as the output part in the corresponding test for RAP. In order for each test to be satisfied, the output of the "box as in the RAP proof" must be .6 or .7, as if the rest of the net did not exist.

(End of Proof)

Fact The proofs in this section require that all tests share only two principal paths. However, it should appear evident to the reader that similar proofs could be given for which only an arbitrarily small fraction of all (leaf) attenuations is not on the principal path of some test.

Refinement from good estimates

Definition 9.2 A *correct* assignment of attenuations for a set of tests is an assignment of attenuations for which all tests are (precisely) satisfied.

Remark In the MIN/MAX case, the error on the output CF (for a given test) is linearly related to the error on the attenuation on the principal path for the test. (Recall from chapter 4 that one needs to consider only one attenuation for each principal path.) In particular, the error on the output CF is less than the error on the attenuation on the principal path, if all CFs and all weights are between 0 and 1. (See Figure 9.9.) This holds even if the error is so large that the principal path with erroneous attenuations is not the one that would be principal if all attenuations were correct: in this case, the error is no more that the error on the attenuation on the path that is principal with erroneous attenuations.

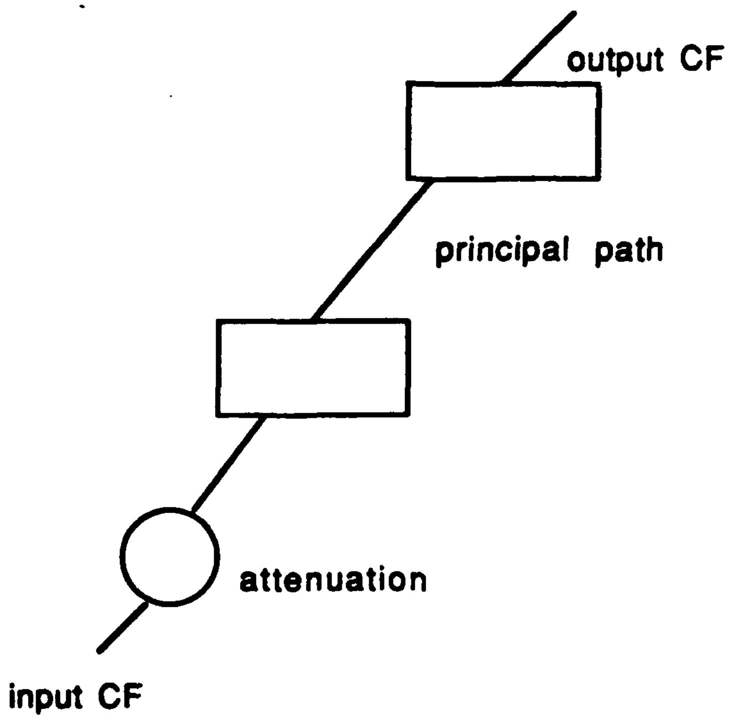


Figure 9.9 Error propagation

In the problems discussed in this section, it is always assumed that a correct assignment of attenuations exists for the tests in the problem instance. This is done in order to simplify presentation of the problems, since the NP-Hardness results that will be proven hold for problems that include instances for which no correct assignment of attenuations exist.

Problem name. Epsilon refinement (ER).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real values at the output of MIN boxes, fan-in to MINs equal to 2, depth 3; a set of tests; a constant epsilon; an assignment of attenuations, each of which is at most epsilon away from the correct one.

Question. Find the correct attenuations.

The decision problem corresponding to ER is introduced now.

Problem name. Epsilon refinement, decision version (ERD).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real values at the output of MIN boxes, fan-in to MINs equal to 2, depth 3; a set of tests; an assignment of "expert-given" attenuations; a constant epsilon (indicated e in the following).

Question. Is there an assignment of attenuations for which all tests are handled correctly, such that each attenuation is at most epsilon away from the expert-given one?

ER is reducible to ERD. Informally, if one could quickly find the correct attenuations, one could quickly answer whether the correct attenuations exist. Therefore, if ERD is NP-Hard, so is ER.

Theorem 9.4. ERD is NP-Hard, for any positive value of epsilon.

Proof.

(This proof is similar to the proof that RA in NP-Complete in section 2 of chapter 5.)

Monotone 3-Conjunctive Normal Form Satisfiability (MSAT) is transformed to ERD. Given an instance of MSAT, i.e. an expression E in monotone 3-conjunctive normal form, the following algorithm will produce in time polynomial in the size of E an instance of ERD such that the Question has answer yes if and only if E is satisfiable.

Let n be the number of distinct variables in E, m the number of clauses in E. (n and m can be obtained in polynomial time from any "reasonable" encoding of E.)

The tree of (the) ERD (instance) has three levels: a MIN box, two MAX boxes under it, attenuators under the MAX boxes. There are 2n+1 attenuators under each MAX box, numbered 1 through 4n+2. (See Figure 9.10.) (The tree has size polynomial in n and therefore in the size of E.)

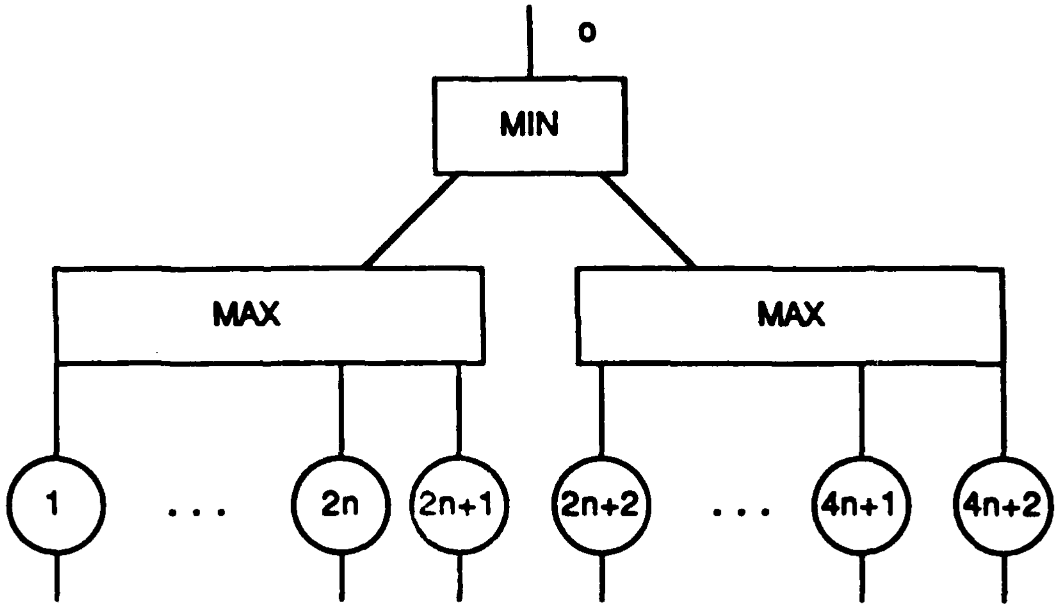


Figure 9.10 Tree of the generic ERD instance

Name the variables in E $x_1, x_2, \ldots, x_n$.

There are 3n+m+1 tests for the ERD instance. All expert-given attenuations in the ERD instance have value a, where e<a<1-e, except for the attenuations in positions 2n+1 and 4n+2, which have value 1.

Let s be a value, such that 0<s<1. Let u=e/2. Let k_1, k_2 be distinct values such that k_i(a+e)>=s(a+u)>=k_i(a-e), i=1,2. (s,k_1,k_2 will be used as CF values in tests.)

One test in the ERD instance has all inputs set to 0 except inputs 2n+1 and 4n+2, which have value s; the output has value s. This test insures that any solution to ERD has these attenuations set to 1. (See Figure 9.11)

1	2	...	2n	2n+1	2n+2	...	4n+1	4n+2	output
0	0	...	0	s	0	...	0	s	s

Figure 9.11 Test of the generic ERD instance

Each clause in E has one test corresponding to it in ERD. This test has all inputs set to 0 except (a) if the clause is positive and contains literals x_{i1}, x_{i2}, x_{i3}, inputs i_1, i_2, i_3 have value s, input 4n+2 has value 1; the output is s(a+u) (b) if the clause is negative and contains literals $-x_{i1}$, x_{i2}, $-x_{i3}$, inputs $2n+1+i_1$, $2n+1+i_2$, $2n+1+i_3$ have value s, input 2n+1 has value 1; the output is s(a+u).

It is a fact that, if a clause with literals x_{i1}, x_{i2}, x_{i3} ($-x_{i1}$, $-x_{i2}$, $-x_{i3}$) is satisfied in isolation, the corresponding test forces all of the attenuations in position i_1, i_2, i_3 (resp. $2n+1+i_1$, $2n+1+i_2$, $2n+i+i_3$) to be at most a+u, and at least one to have value a+u.

Each variable in E has three tests corresponding to it in ERD. (These tests are designed to force exactly one of the attenuations in positions i and 2n+1+i to be a and the other one to be different from a, but within e of it. Recall that, analogously, in the proof of RA, pairs of attenuations could take values (0,1) or (1,0).)

The first test for variable x_i has all inputs set to 0 except inputs i and 2n+1+i, which have value s; the output is s*a. (See Figure 9.12.) It is a fact that such a test is satisfied if and only if one of the attenuations in positions i and 2n+1+i has value a and the other has value greater than or equal to a.

1	...	i-1	i	i+1	...	2n+i	2n+1+i	2n+2+i	...	4n+2	output
0	...	0	s	0	...	0	s	0	...	0	s*a

Figure 9.12 Test of the generic ERD instance

The second and third test are similar to each other; the only difference is that k_2 is used in the third test where k_1 is used in the second one. Only the second test will be now described. This test has all inputs set to 0 except inputs i and 2n+1+i, which has value s, input n+i, which has value k_1, and input 2n+1+n+i, which also has value k_1. The output of each test is s(a+u). (See Figure 9.13.)

It is a fact that each such pair of tests is satisfied if and only if

(a) the attenuation in position i has value a+u, and the attenuation in position 2n+1+i has value no less than a+u, or the attenuation in position 2n+1+n+i has value no less than $s(a+u)/\min(k_1,k_2)$, or

(b) the attenuation in position 2n+1+i has value a+u, and the attenuation in position i has value no less than a+u, or the attenuation in position n+i has value no less than $s(a+u)/\min(k_1,k_2)$.

Recall from section 3 of chapter 4 that a principal input is the input to a principal path, that is to a path whose attenuated input is equal to the output. Note that k_1 and k_2 have been chosen in such a way that $s(a+u)/\min(k_1,k_2)$ is within e of a and that $k_1 <> k_2$, which insures that the principal input for the two tests is neither n+i nor 2n+1+n+i, by an argument similar to that used in the NP-Hardness proof for AS (section 2, chapter 5).

1	...	i	...	n+i	...	2n+1+i	...	2n+1+n+i	...	4n+2	output
0	0	s	0	0	0	s	0	0		0	s*a
0	0	s	0 0	k_1	0 0	s	0 0	k_1	0	0	s(a+u)
0	0	s	0 0	k_2	0 0	s	0 0	k_2	0	0	s(a+u)

Figure 9.13 Tests of the generic ERD instance

Lemma The variable tests for variable x_i are satisfied if and only if the pair of attenuations in position i and 2n+1+i is either (a,a+u) or (a+u,a).

Proof The first test is satisfied if and only if one of these attenuations has value a and the other has no smaller value; the second and third test are satisfied only if one of the attenuations has value a+u.

(End of proof of lemma.)

It will now be shown that E has a model if and only if the ERD instance built using the algorithm described above is a yes-instance.

(a) E has a model implies that ERD is a yes-instance.

Assume that E has a model.

Since E is in CNF, each of the clauses is satisfied.

We provide a mapping from the truth values of the variables in E to attenuations that provide a solution to ERD. For each variable x_i,

(a) if x_i=T in the model, let a_i (the i-th attenuation) be a, $a_{2n+1+i}=a+u$, $a_{n+i}=s(a+u)/\min(k_1,k_2)$, $a_{2n+1+n+i}=s*a/\min(k_1,k_2)$, $a_{2n+1}=a_{4n+2}=1$.

(b) if x_i=F in the model, let $a_{2n+1+i}=a$, $a_i=a+u$, $a_{2n+1+n+i}=s(a+u)/\min(k_1,k_2)$, $a_{n+i}=s*a/\min(k_1,k_2)$, $a_{2n+1}=a_{4n+2}=1$.

It is an easy, although tedious, task to verify that all tests in ERD are satisfied by this assignment of attenuations; we leave it to the reader. Moreover, u, k_1, and k_2 have been defined in such a way that all attenuations are within e of the expert-given ones, proving that the ERD instance built according to the previously described algorithm is a yes-instance if E has a model.

(b) ERD is a yes-instance implies that E has a model.

Assume that ERD is a yes-instance, i.e., that there is an assignment of attenuations for which all tests are satisfied, and this assignment is within e of the expert-given one.

Consider the pair of attenuations (a_i,a_{2n+1+i}). Recall that u is defined as e/2, and e is given as part of the ERD instance. We claim that the mapping

if $(a_i,a_{2n+1+i})=(a,a+u)$ then x_i=T
if $(a_i,a_{2n+1+i})=(a+u,a)$ then x_i=F

is a model for E.

Firstly, note that, by the Lemma, the mapping is a total function from the space of variable names to {T,F} (therefore, an interpretation). (This is the use made of variable tests in this proof.)

Secondly, note that an interpretation for E is a model if and only if each clause of E is true under the interpretation, since E is in CNF. Clearly, if all tests are satisfied, then all clause tests are satisfied. Consider generic test T_r corresponding to clause C_r, 1<=r<=m. If C_r is a positive clause, $C_r=(x_{i1}, x_{i2}, x_{i3})$, T_r is satisfied if and only if at least one of the following equalities holds:

$(a_{i1},a_{2n+1+i1})=(a,a+u)$, $(a_{i1},a_{2n+1+i2})=(a,a+u)$,
$(a_{i3},a_{2n+i+i3})=(a,a+u)$.

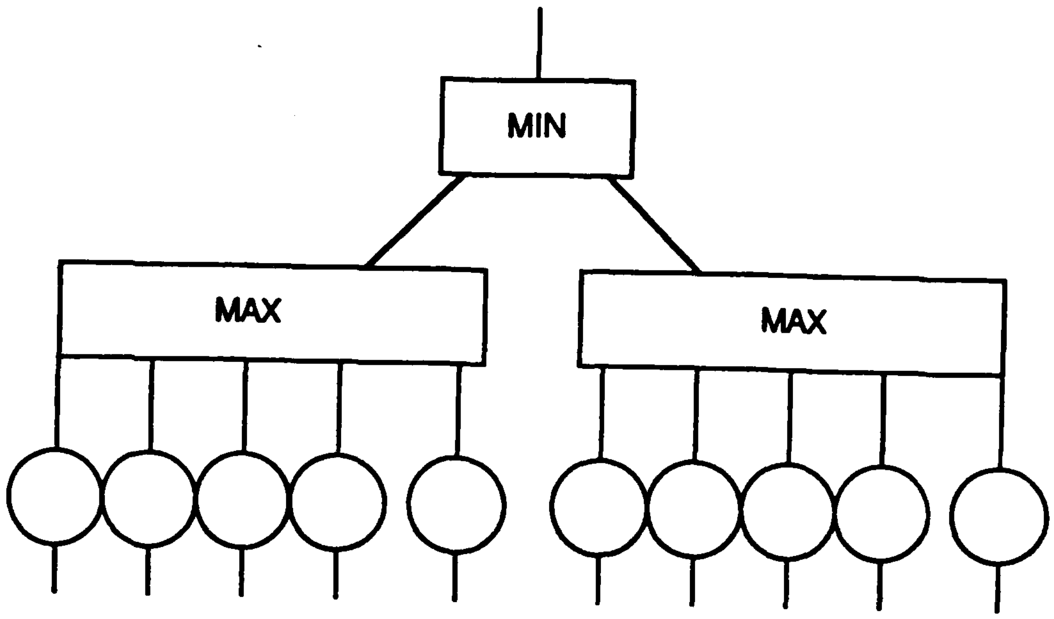
Therefore, since the test is satisfied, our mapping will set one of x_{i1}, x_{i2}, x_{i3} to T. Since C_r is a disjunctive clause, it is true in our interpretation. The case in which the clause is negative is analogous: (a+u,a) and F should be substituted to (a,a+u) and T, respectively. Therefore, each clause is true in the interpretation we have defined; the claim has been proven.

(End of proof of theorem.)

Example 9.5

An example of the mapping from MSAT to ERD is shown in Figure 9.14. Only 2-literal clauses are used for clarity of example.

$E = (x_1 v x_2) \& (-x_1 v -x_2)$



											output
	0	0	0	0	s	0	0	0	0	s	s
$x_1 v x_2$	s	s	0	0	0	0	0	0	0	1	s(a+u)
$-x_1 v -x_2$	0	0	0	0	1	s	s	0	0	0	s(a+u)
x_1	s	0	0	0	0	s	0	0	0	0	s*a
	s	0	k_1	0	0	s	0	k_1	0	0	s(a+u)
	s	0	k_2	0	0	s	0	k_2	0	0	s(a+u)
x_2	0	s	0	0	0	0	s	0	0	0	s*a
	0	s	0	k_1	0	0	s	0	k_1	0	s(a+u)
	0	s	0	k_2	0	0	s	0	k_2	0	s(a+u)

Figure 9.14 ERD instance corresponding to $(x_1 v x_2) \& (-x_1 v -x_2)$

(End of example)

A natural question to ask, given the negative result just obtained, is whether it is possible to improve on a (slightly) incorrect assignment of attenuations. This problem can be formalized as follows.

Definition 9.3 Given an inference net, the error on a test is the difference between the output part of the test and the output CF (CFs) obtained by propagating the input part of the test through the tree (graph).

Problem name. Approximate epsilon refinement (AER).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with real values at the output of MIN boxes, fan-in to MINs set to 2, depth 3; a set of tests; a constant e; an assignment of (expert-given) attenuations, each of which is at most e away from the correct one; a constant q.

Question. Find an assignment of attenuations
(a) within e of the expert-given attenuations
(b) for which the error on each test is no greater than using the expert-given attenuations
(c) for which the error on at least one test is q less than using the expert-given attenuations, or a non-zero error is reduced to zero.

Theorem 9.5 AER is NP-Hard, for arbitrarily small e and q.

Proof If this were not the case, the following algorithm would solve ER in polynomial time. This would contradict the NP-Hardness of ER, proven previously.

Let k=e/q. Let n be the number of attenuations.

Repeatedly use the polynomial-time algorithm to solve AER, using the solution to an iteration as the starting point for the next iteration, until the correct attenuations are obtained.

Since each time the algorithm is applied at least one error is decreased by q or a correct attenuation is found (and therefore at least one attenuation gets q closer to the correct one or is set to the correct value), in at most n+nk iterations all errors are 0, i.e., ER is solved.

(End of proof.)

A fast algorithm

In this section, the NP-Hardness proofs presented in the previous section will be exploited in order to gain insight into the reason why epsilon refinement and approximate epsilon refinement are hard.

Definition 9.4 A choice box is a function whose output is equal to (at least) one of the inputs. (Of course, the output is equal to more than one input only if at least two inputs are equal.)

For example, MIN and MAX are choice boxes.

Definition 9.5 A winner at a choice box is the input value that is equal to the output value. (In the case of output values that are equal to more than one input value, the winner is chosen arbitrarily among these inputs.)

Definition 9.6 A setting of winners for an inference net is an assignment of winners to all choice boxes in the inference net.

From now on in this section, only inference nets with 2-input boxes will be considered. Definitions 9.7 and 9.8 below may be extended to nets with arbitrarily large boxes. However, this extension is not required to obtain the general results that will be shown, since there is a fast algorithm to convert a net with multiple-input boxes into a net with only 2-input boxes, as shown in the following example.

Example 9.6

The inference tree shown in Figure 9.15 is given. It can be transformed into the tree shown in Figure 9.16, because it is possible to set all attenuations but leaf attenuations to value 1, as demonstrated in Chapter 4. Finally, each n-input choice box can be transformed into (n-1) suitably interconnected 2-input choice boxes as shown in Figure 9.17, without introducing additional attenuations. Given an assignment of attenuations for a_1 through a_{14}, and an input part for the trees in Figure 9.16 and 9.17, the same CF will flow on line o for both trees. If necessary, the algorithm described in section 4 of chapter 4 can be used to redistribute attenuations a_1 through a_{14} into attenuations x_1 through x_{16}.

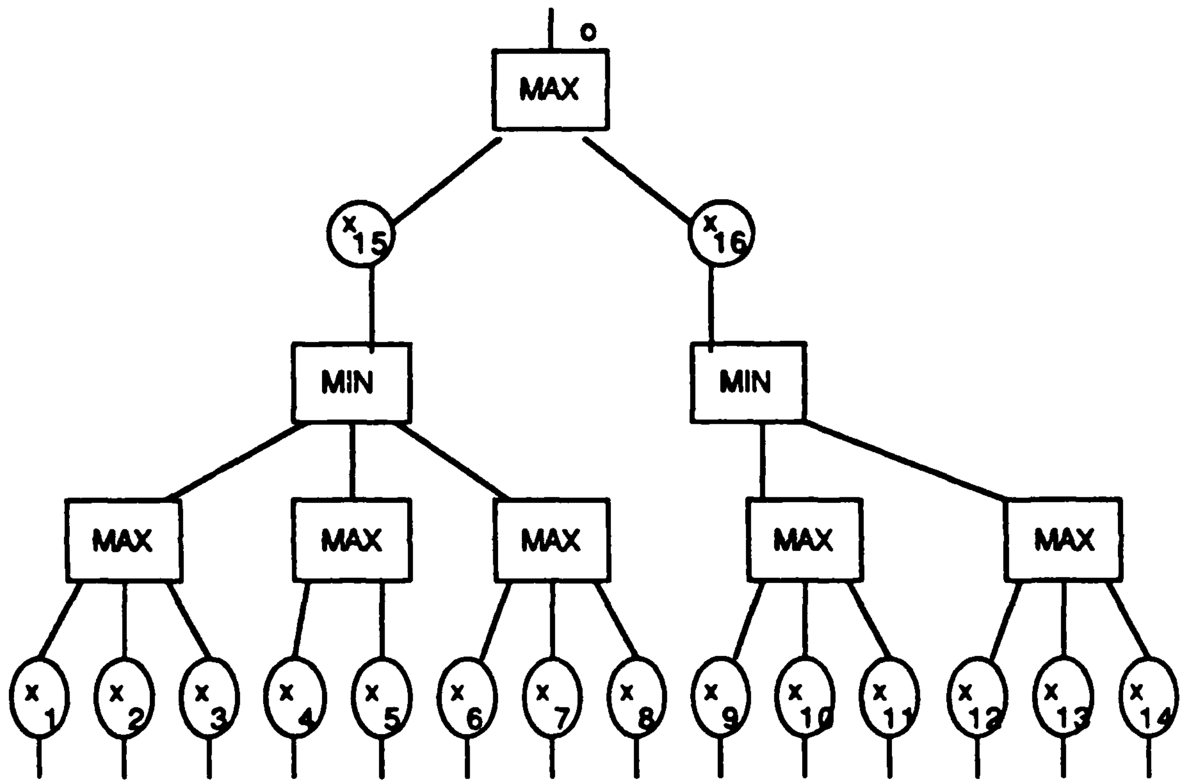


Figure 9.15 Converting n-input boxes into 2-input boxes (initial phase)

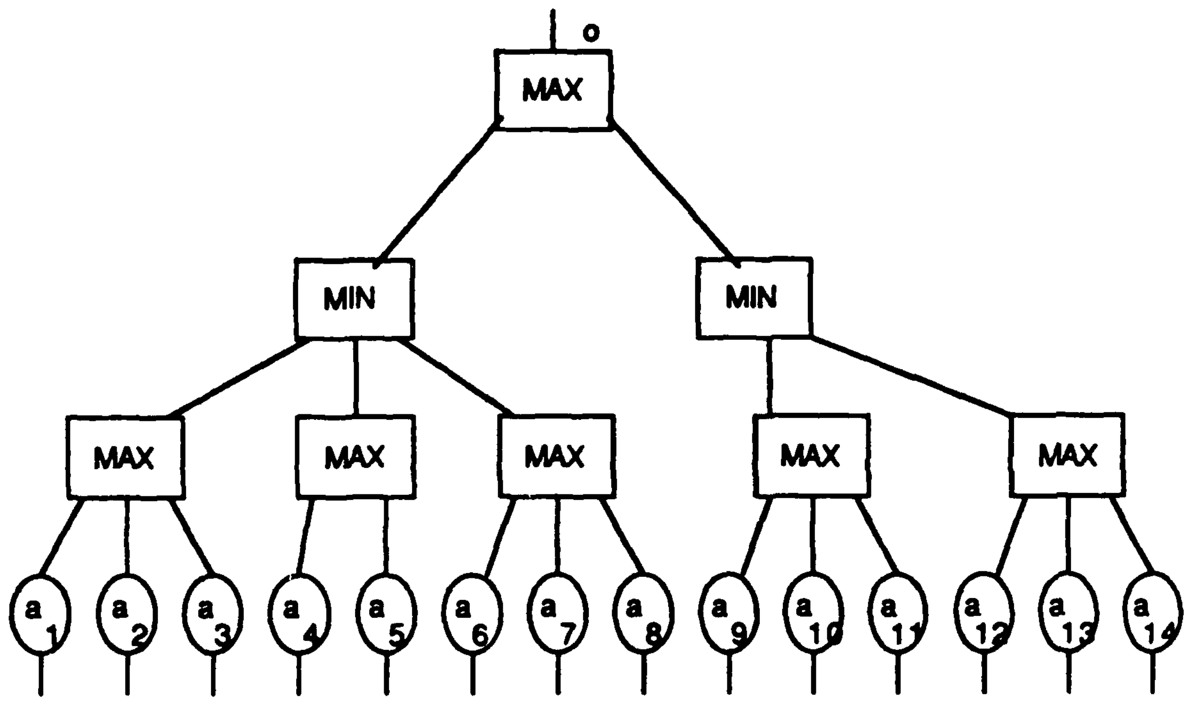


Figure 9.16 Converting n-input boxes into 2-input boxes (intermediate phase)

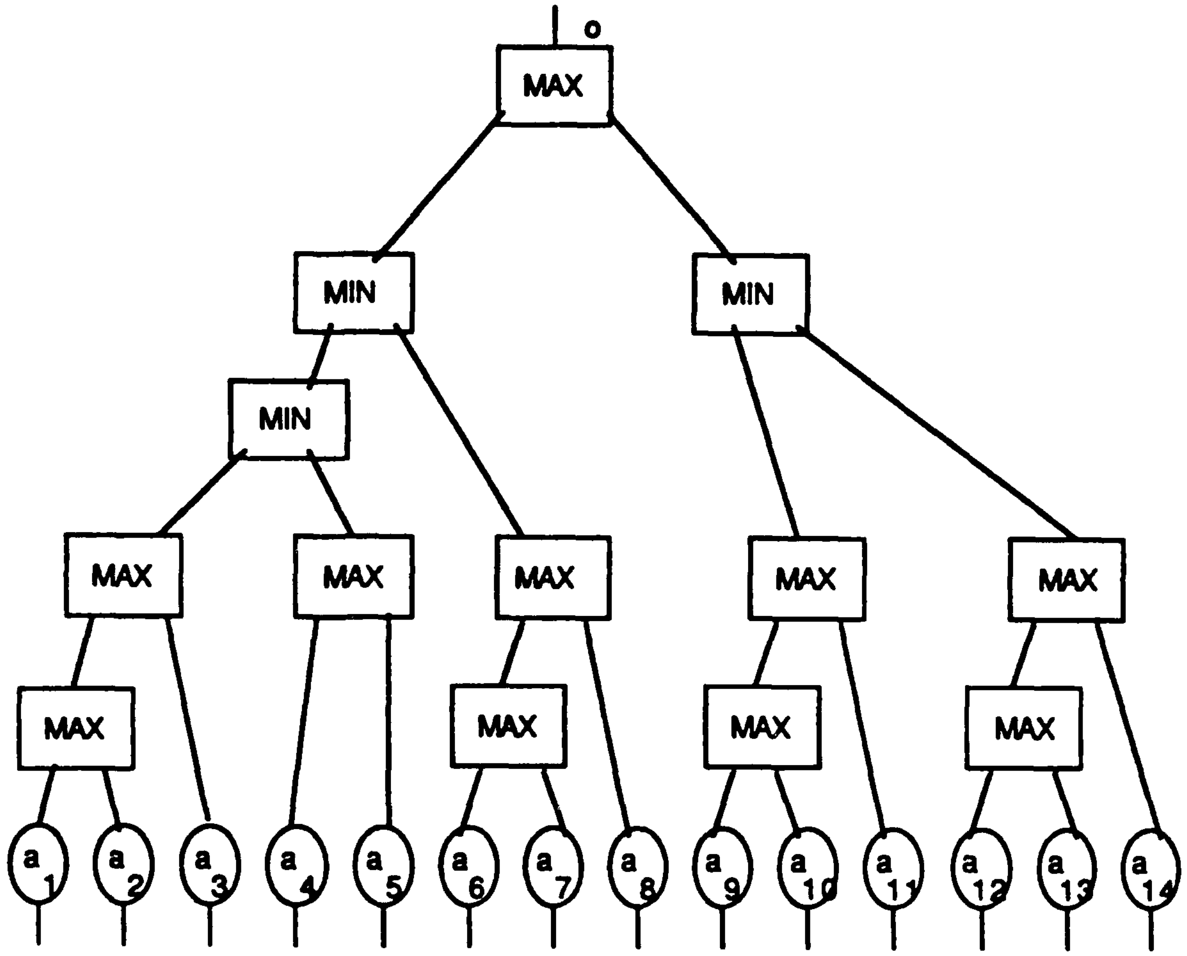


Figure 9.17 Converting n-input boxes into 2-input boxes (final phase)

(End of example)

Definition 9.7 A loser at a 2-input choice box is the input value that is not equal to the output value. In the case of a tie, the loser is chosen to be the other value with respect to the arbitrarily chosen winner.

Definition 9.8 A switch setting for an inference net with 2-input boxes is a setting of winners (or, equivalently, of losers) for the inference net.

Note that the NP-Hardness proofs in the previous section do not carry through if the problem instance is augmented to include as input a switch setting for the tree. This observation leads to the definition of the following search problem. (The reader can imagine what the corresponding decision version is.)

Problem name: Attenuation synthesis, no-switch case (ASN) (Search version).

Problem instance: A tree with 2-input MIN and MAX choice boxes, multiplicative attenuators with real values at the output of MIN boxes; a set of tests TS=$\{T_1,\ldots,T_j\}$; a switch setting for each test in TS.

Question: Find an assignment of attenuations compatible with the switch setting, such that all tests are satisfied, if such an assignment exists.

Algorithm 9.1

(The algorithm has as input an instance of ASN and returns an assignment of attenuations, as requested in the Question for ASN, if such as assignment exists. It ends in failure at step 2, if such an assignment does not exist. Let n be the number of leaf attenuations, defined in section 2 of chapter 4, in the tree.)

1. For each test, T_i, and each choice box, set up a 2-variable linear inequality as follows: let $a_{i1}*i_{i1}$ be the winner at the choice box; let $a_{i2}*i_{i2}$ be the loser. If the box is a MAX box, let the inequality be $a_{i1}*i_{i1}>=a_{i2}*i_{i2}$. If the box is a MIN box, let the inequality be $a_{i1}*i_{i1}<=a_{i2}*i_{i2}$. Add 2n inequalities $a_i>=0$, $a<=1$, i in $\{1,\ldots,n\}$. (There are at most 3n-1 inequalities for each test, since there are n-1 choice boxes in a complete tree with n leaf attenuations, i.e., n/2 leaf choice boxes.)

2. Solve the system of inequalities obtained in Step 1.

(End of algorithm)

Theorem 9.6 Let n be the number of leaf attenuations (defined in section 2 of chapter 4) in the tree. Let m=n*j. Algorithm 9.1 solves ASN in $O(m \log mn^3)$. (Recall that j is the number of tests in the ASN instance.)

Proof

It is clear by the construction in phase 1 of the algorithm that a solution to the system of linear inequalities is a solution to ASN and that no solution to the ASN instance exists if the system of linear inequalities is unsatisfiable.

Khachiyan [1979] presents an algorithm to solve a system of linear inequalities in polynomial time. However, faster algorithms are known for the case in which only 2 variables per inequality are present. Some of these algorithms are particularly interesting, because they require polynomial run time independently of the encoding scheme used to represent the coefficients of the linear inequalities, whereas Khachiyan's algorithm requires the use of a particular such scheme, the binary encoding scheme. Johnson [1983] and Megiddo [1982] discuss the binary encoding scheme and real arithmetic model. In particular, Megiddo proves that there exist encoding schemes that are as efficient as the binary encoding scheme, but with respect to which Khachiyan's algorithm for linear programming is exponential. Johnson describes the real arithmetic model for linear programming as the model in which each arithmetic operation has unit cost and run-time is expressed as a function of the number of variables and constraints. Megiddo [1983] presents an algorithm to solve a system of linear inequalities with m=n*j inequalities, n variables, and at most 2 variables per inequality in time $O(m \log mn^3)$. Since constructing the system as done in phase 1 of the algorithm takes only time O(m), the claim holds.

(End of theorem.)

The previous theorem indicates a way to exploit expert-given attenuations in refining attenuations. The expert-given attenuations to be refined should be used to compute winners at each box for each test. The inequalities are set up assuming that winners do not change when attenuations are changed from the expert-given estimates to the correct ones. If the expert-given attenuations are close enough to the correct ones to be accurate predictors of winners, this technique is successful.

The following examples illustrates the procedure informally described above; it is expected that the reader will easily succeed in filling in the missing details.

<u>Example 9.7</u>

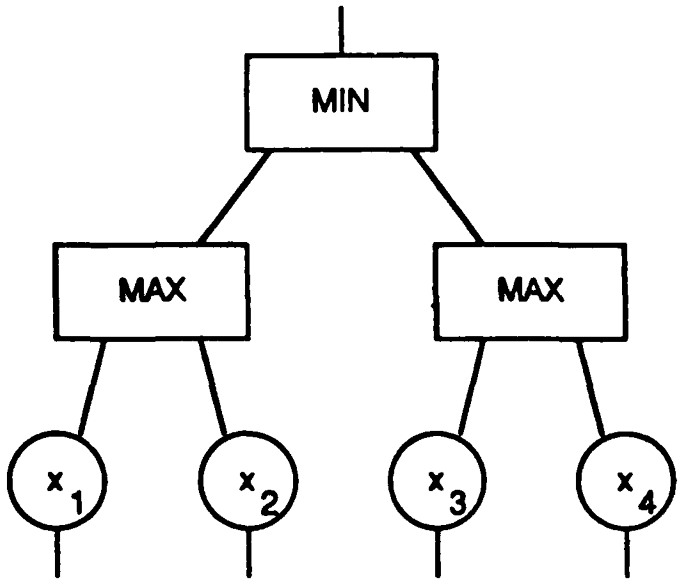


Figure 9.18 Inference tree for example 9.7

Consider the inference tree in Figure 9.18. The expert-given attenuations are $\mathbf{x}^E=(.5,1,.4,.4)$. The test set consists of the single test $T_1=((.6,.5,.8,.2),.1)$.

Assuming that winners do not change with respect to the ones given by the expert-given attenuations, one has the following system of 7 inequalities and one equality:

$.8x_3-.1=0$ (This corresponds to the principal path for T_1.)
$.6x_1 <= .5x_2$
$.8x_3 >= .2x_4$
$.8x_3 <= .5x_2$
$0<=x_i<=1,\ i=1,..,4.$

This system has an infinite number of solutions, for example: $\mathbf{x}=(.5,1,1/8,.4)$.

(End of example)

<u>Example 9.8</u>

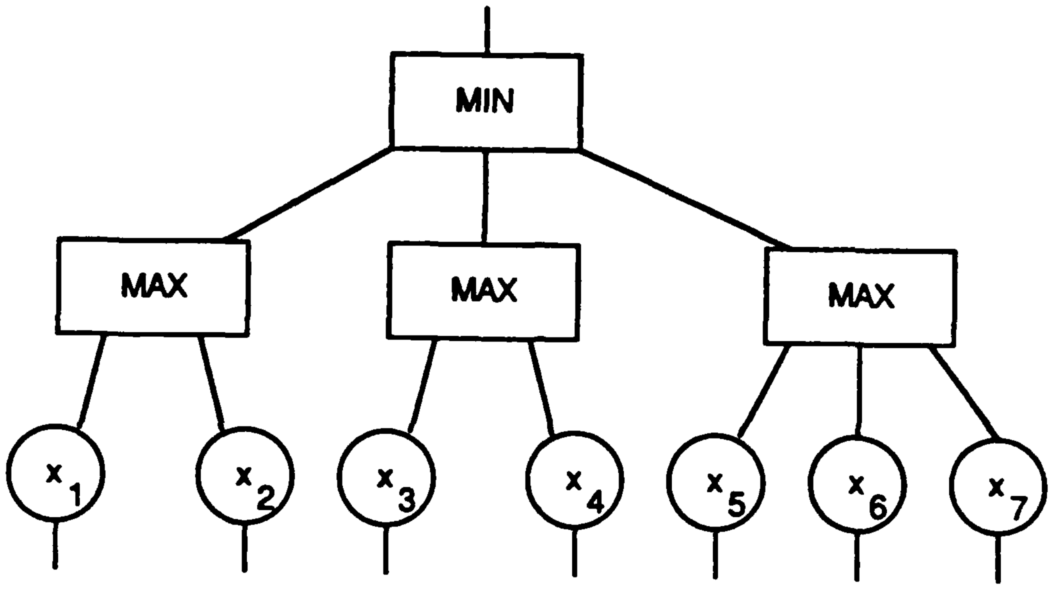


Figure 9.19 Inference tree for example 9.8

Consider the inference tree in Figure 9.19. The expert-given attenuations are $\mathbf{x}^E = (.5,1,.4,.4,.6,.4,.5)$. The test set consists of the single test $T_1 = ((.6,.5,.8,.2,.7,.6,.9),.4)$. Assuming that winners do not change with respect to the ones given by the expert-given attenuations, one has the following system of 13 inequalities and one equality:

$.8x_3-.1=0$ (This corresponds to the principal path for T_1.)
$.6x_1 <= .5x_2$
$.8x_3 >= .2x_4$
$.8x_3 <= .5x_2$
$.7x_5 >= .6x_6$
$.7x_5 <= .9x_7$
$.8x_3 <= .9x_7$
$0<=x_i<=1, \; i=1,\ldots,7$

A solution to this system is, for example,
$\mathbf{x}=(.6,.5,.5,.4,.6,.4,.5)$.

(End of example)

It can be noted that a sufficient and (in the worst case over distributions of input values) necessary condition for the expert-given attenuations to be good winner predictors is that each (attenuation*input) product be 2e away from any other, where ϵ is the maximum error on expert-given estimates.

What should be done when the estimates for winners obtained from the expert-given attenuations are not correct? And how should one provide for the possibility that there is no solution to satisfy the given tests, possibly because of noise in the tests themselves? These questions will be addressed in the following section.

Iterative algorithms

Several algorithms will be now presented. These algorithms all iterate a basic step, which is different for each algorithm. Each step consists of the selection of a path in the tree (graph) and of the adjustment of the weight (weights) on that path. Unless specified otherwise, we will present the algorithms that operate on trees.

Algorithm 9.2

```
Until convergence for each test
             find a principal path (break ties
                    arbitrarily) modify the attenuation
                    on the PP in such a way that PP's
                    output is test's output
```

This algorithm does not converge when a solution does not exist. Just picture the case in which the tree is a chain and two tests are given with the same input but different outputs. Of course, because of the NP-completeness result shown in section 2 of chapter 5, it must be that the algorithm is slow or non-convergent on some cases for which a solution exists. A measure of the "quality" of these algorithms is the number and nature of the cases in which they either are slow or do not converge at all. We also know that there are no fast algorithms that can provide a (suitably defined) approximation in all cases, so that we cannot expect any iterative algorithm to converge (in a suitable sense) in a short time in all cases. In this respect, the first algorithm does not seem very good. Here are two examples in which this algorithm does not converge at all, but a solution exists.

Example 9.9

The first example involves two tests sharing the same incorrect principal path. Since no step ever causes a change in principal path, the algorithm never terminates.

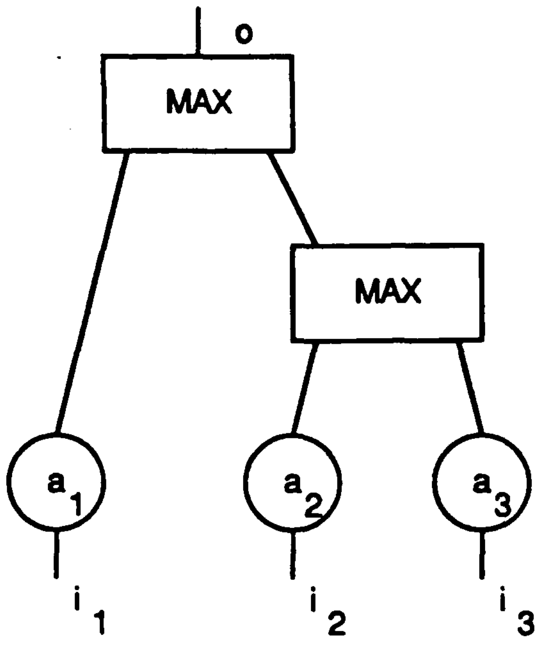


Figure 9.20 A tree on which Algorithm 9.2 fails

The inference tree is shown in Figure 9.20. The expert-given attenuations are $\underline{a}^E=(.7,.4,.5)$. The two tests are $T_1=((1,.9,.5),.8)$ and $T_2=((1,.5,.7),.6)$. The value of attenuation a_1 would indefinitely switch between .8 and .6, while there exists an infinity of solutions a=(t,8/9,6/7), where 0<=t<=.6.

(End of example)

<u>Example 9.10</u>

Three tests are involved in the second example. This example shows that it is possible for the algorithm to loop infinitely on a triplet of tests, even when no pair which is a subset of the triplet would cause nontermination.

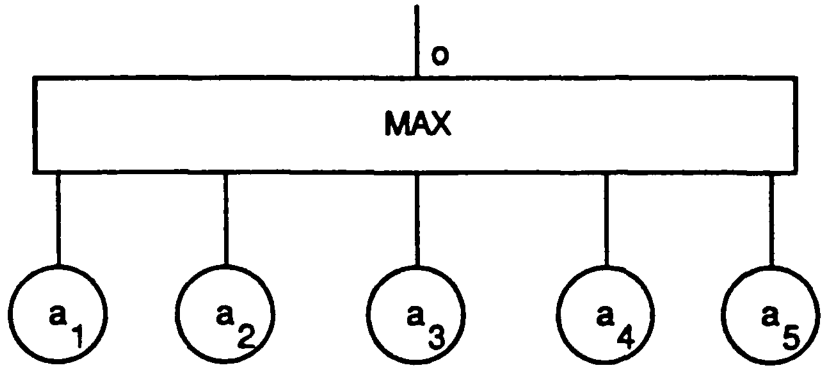


Figure 9.21 A tree on which Algorithm 9.2 fails

The inference tree is shown in Figure 9.21. The expert-given attenuations are $\underline{a}^E=(.7,.51,.1,.1,.1)$. The three tests are $T_1=((1,0,1,0,0),.75)$, $T_2=((0,1,0,1,0),.4)$, $T_3=((1,1,0,0,1),.6)$. Algorithm 9.2 would never terminate on this example: a_1 would be set to .75, and a_2 would indefinitely switch between .4 and .6, while there exists a class of solutions, including (0,0,.75,.4,.6).

(End of example)

One would like to say that in these examples and in all the cases in which Algorithm 9.2 does not converge the expert-given estimate for the weights is a "bad estimate," but no simple characterization of this has been found.

The following two algorithms are based on Rada's work [Rada, 1985]. In order to state these algorithms, one needs to define some properties of attenuators: TooMuch, Perfect and TooLittle. For each weight, TooMuch is the number of tests for which the outcome of the system is greater than the desired output, <u>and the weight is on a principal path</u>. TooLittle and Perfect mean just what one would expect, given the definition of TooMuch. Because of the qualification that the weight be on a principal path,
TooLittle + Perfect + TooMuch is not necessarily equal to the number of tests for each attenuation. Each iteration in

Rada's algorithms is more complicated than in Algorithm 9.2. It consists of running all cases on the system and computing the values of TooMuch, Perfect, and TooLittle for each attenuation. The attenuation to be modified during the current iteration is chosen according to these values, in different ways for each of the two algorithms.

<u>Algorithm 9.3</u>

```
Until convergence compute TooMuch, Perfect and TooLittle
     for each attenuation
             if TooMuch - TooLittle - Perfect > 0 then
                     decrement the attenuation
             else if TooMuch - TooLittle - Perfect < 0
                     then increment the attenuation
```

<u>Algorithm 9.4</u>

```
Until convergence compute TooMuch, Perfect and TooLittle
     if there is an attenuation for which TooMuch > 0,
     TooLittle=Perfect=0
             then decrement it else increment the
     attenuation with the lowest value of
             Perfect+TooLittle
```

It is easy to find examples for which Algorithm 9.3 fails: in fact, it fails on both Examples 9.8 and 9.9 (on which Algorithm 9.2 also fails). Here is an example of a case in which Algorithm 9.4 fails.

<u>Example 9.11</u>

In this case, the algorithm terminates without finding a solution, whereas a solution does exist.

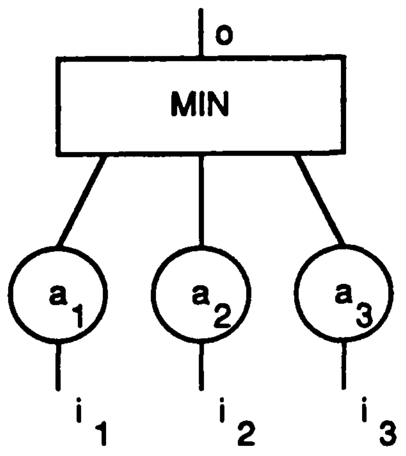


Figure 9.22 A tree on which Algorithm 9.4 fails

The inference tree is shown in Figure 9.22. The expert-given attenuations are $\underline{a}^E=(.7,.4,.5)$. The two tests are $T_1=((1,.9,.1),.75)$ and $T_2=((1,.8.,8),.6)$. (In the following, p stands for Perfect, tl for TooLittle, tm for TooMuch.) On the first iteration, $p[a_1]=tl[a_1]=0$, $p[a_2]=tl[a_2]=2$, $p[a_3]=tl[a_3]=0$. Weights a_1 and a_3 get incremented to 1, because their p+tl is lowest. a_2 gets incremented next. It is the only attenuation that can get incremented; however it is incremented, a situation in which $tm[a_3]>0$ can never be reached, and therefore a_3 cannot be decremented. This insures that a solution cannot be found, whereas solutions like (1,1,3/4) do exist.

(End of example)

The algorithm presented here all fail, because they do not switch winners--in particular, they do not switch winners on principal paths. In particular, consider Algorithm 9.2. For the example on which this algorithm fails, a_2 should be on the principal path for T_1 and a_3 should be on the principal path for T_2, but a_1 remains principal for both tests; again, the algorithm is unable to force a switch (of winners) for the given tests. Algorithm 9.3 works just like Algorithm 9.2 on the example considered.

Algorithm 9.4, on the example given above, is unable to force a switch between a_2 and a_3.

Therefore, these iterative algorithms are poor in that, while using the estimates to predict the "next switch setting," are unable to consider all switch settings, even for simple examples.

An algorithm that finds a solution if it exists must search the space of switch settings in such a way that all of them are eventually tried. The expert-given attenuations provide a starting point for this enumeration. The best ordering, given a suitable model of expert error, is to try first all switch settings in which exactly one switch is set differently from the expert-given one, then all switch settings are set differently, and so on; within each class of switch settings, the first switch setting to be changed is that for which the margin of victory is the smallest.

Different models of expert error will lead to different enumeration orderings for switch settings. Moreover, standard techniques could be used to reduce the number of settings to be tested. This is an area for further research, possibly of an empirical nature.

CHAPTER TEN

CONCLUSION

Introduction

This last chapter contains two parts: a personal assessment of the results contained in the thesis and a discussion of open problems.

Assessment of results

I will not summarize the results here, since this was already done in chapter 1. Instead, I will try to give a personal assessment of the results obtained.

Here is a list of lessons, with an indication of the chapters or sections where the related results are discussed:

(1) Perfect experts are (quantifiably) better than cases, or, if one prefers, complete information is better than incomplete information to solve the synthesis problem. (Compare chapters 4 and 5.)

(2) Providing information on intermediate hypothesis (quantifiably) simplifies the synthesis problem and makes it computationally tractable. (However, it places a much heavier burden on the team composed of the expert and the knowledge engineer.) (Compare section 8.3 with 5.2.)

(3) Attenuations that are not closed under composition are difficult to synthesize. (Chapter 6)

(4) Synthesis from cases is intractable, even for simple network topologies. (Chapter 5)

(5) Refinement from cases is difficult, even for simple network topologies and estimates that are very close to correct weights. (Chapter 9)

(6) Synthesis from perfect experts is easy for simple topologies (chapter 4), but it becomes intractable for more complex topologies. (Chapter 7)

It has been said that "everything we want to do is either NP-Complete or undecidable, i.e everything interesting is too hard" [Brachman, 1986]. To some extent, this thesis confirms the "folk theorem" so clearly stated by Brachman. Still, it is my judgement that the field of expert systems is badly in need of formalizations that make it possible to apply established techniques of "mainstream" Computer Science.

The main lesson that I derive from this work is that it is unlikely that large expert systems can be built using MYCIN-like rules. (Here, "large" may mean over two or three thousand rules.) Users of expert system shells hope that practical tools will soon be developed "to refine or add to the knowledge base as the intelligent system has new experiences" [Hafner, 1986]. This expectation is unfounded. It seems more sensible to pursue, as done in recent commercial developments [Richer, 1986], the approach of encouraging modularity and the use of multiple formalisms to keep the size of individual rule bases down.

Directions for further research

There are several variants of the synthesis and refinement problem that have not been considered in this thesis, but which seem to be of practical interest.

The first variant may be called refinement with rigidities. Some rule attenuations are considered less tunable than others, because of explicit indication by the expert, of past experience with those rules (perhaps in other rule bases), or to simplify refinement. This last possibility is especially intriguing. Refinement could be confined to a subset of all the rules. However, the results in chapter 7 show that, at least in some instances, even the determination of one rule attenuation is very difficult.

The second variant may be called refinement with noisy tests. If the assumption that the test cases are all correct does not hold, one has to account for the possibility of noisy tests. This is especially possible if the tests are collected automatically, rather than from cases screened or designed by an expert. Two approaches to the solution of this problem are: either satisfy only a subset of the test cases, or compromise on exact performance on all tests. In this second approach, one would minimize both the distance between the expert-given attenuations and the refined attenuations and the distance between the goal CFs as given in the test cases and as concluded by the system. There should be good reasons to conclude that some or all tests are noisy, because the inability of the system to satisfy all tests correctly may be due to errors in the structure of the rules, rather than to noise in the tests. See [Saitta, 1984], esp. p.102 for some additional discussion. The first approach will be discussed in one of the sections on open problems, entitled "maximum compatible set of tests."

The three sections that follow describe three open problems and contain some initial results. The first section, "truncated multiplication," is probably the least interesting for applications. I suspect that it may provide a challenge to a researcher in number theory. The problem described there was first attacked by Albert Nigrin [1984].

Open problems: truncated multiplication

Truncated multiplication is a special case of function which is not closed under composition. It was shown in Chapter 6 that the synthesis of attenuations which are not closed under composition in an NP-Hard problem (even for chains), but this, of course, does not imply that the synthesis of attenuations is NP-Hard when a specific class of functions that are not closed under composition is chosen as the class of attenuation functions.

Definition 10.1 A truncated multiplication of order m is a function f: A -> A, such that f(n) = floor(x*n), 0 <= n <= m, 0 <= x <= 1.

Example 10.1

All the truncated multiplications of order 4 are shown, in tabular form, in Figure 10.1.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
4	4	3	2	2	1	1	0
3	3	2	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	0	0	0	0	0	0

Figure 10.1 Truncated multiplications of order 4

(End of example)

Note that there are as many truncated multiplications of order m as there are Farey fractions of order m. Therefore there are approximately $.3*m^2$ truncated multiplications of order m [Hardy and Wright, 1979, p.268]. Each truncated multiplication corresponds to a Farey fraction, in the sense that $f_i(n)$ = floor(F_i*n), where F_i is the ith Farey fraction. For example, for truncated multiplications of order 4, $f_2(n)$ = floor((3/4)*n).

The synthesis problem can be stated as follows.

Instance: An integer m, a finite function h: A -> A, where A = {0,1,...,m}.

Question: Is there a composition of truncated multiplications of order m that is equal to h?

<u>Example 10.2</u>

A particular instance of this problem is the following. Is there a composition of truncated multiplications of order 4 that is equal

```
                 4 -> 2
                 3 -> 1
to      h =      2 -> 0     ?
                 1 -> 0
                 0 -> 0
```

Recall that the truncated multiplications of order 4 are given in the table example 10.1. Note that h is different from each of f_1 through f_7. Also, the answer is "yes," since h is equal to the composition of f_2 with itself.

(End of example)

One could wonder why truncated multiplication is an interesting choice for attenuations. Of course, one could just answer that it is as good a choice of attenuators as any, but the motivation for studying it is that it can be considered as a simulation of real multiplication on a computer: computers only approximate real multiplication and it can be argued that they actually perform truncated multiplication. It must be noted, though, that the error due to rounding is at worst linear when numbers are multiplied. In fact, techniques exist to control errors in a series of multiplications [Kulisch and Miranker, 1981; 1983].

It would be interesting to know whether the truncated multiplication problem is NP-Hard. One would expect that the truncated multiplication problem would have been studied and possibly solved in the context of computer arithmetic, so the author posted a note on Usenet's "net.math" newsgroup, expecting some pointer to the published literature. Several people replied, from places as far away as Carnegie-Mellon University and the CWI (Center for Applied Mathematics) in Amsterdam, but no solutions were known or found. However, Charles Simmons from Dartmouth came up with five conditions, which he conjectured to be necessary and sufficient for functions that are composed of truncated multiplications. Here is the conjecture: A function h is the composition of truncated multiplications if and only if:

(1) $h(0) = 0$
(2) $h(1) = 0$ (unless h is the identity function)
(3) if $n >= 2$ then $h(n) <= n-2$ (unless h is a truncated multiplication)
(4) $h(n+1) <= h(n) + 1$ (the function cannot grow too quickly)
(5) $h(a+b) >= h(a) + h(b)$ (the function cannot grow too slowly).

Simmons showed that these conditions are necessary. (All proofs are by induction.) Unfortunately, the conjecture that the conditions are also sufficient is false. A counterexample is that the following function, which passes all five conditions, is not a composition of truncated multiplications (of order 10):

$$h = \begin{array}{l} 10 \to 7 \\ 9 \to 6 \\ 8 \to 5 \\ 7 \to 4 \\ 6 \to 3 \\ 5 \to 2 \\ 4 \to 2 \\ 3 \to 1 \\ 2 \to 0 \\ 1 \to 0 \\ 0 \to 0 \end{array} .$$

This can be shown by branch and bound search.

<u>Open problems: maximum compatible set of tests</u>

Consider the situation in which we are satisfied with an expert system that handles correctly only a subset of a given set of tests. There are several possible reasons why this may be acceptable: for example, it may be known that a fraction of the tests is incorrect ("noisy") or it may be important to have a small number of rules, even at the price of inaccuracy, for reasons of efficiency. In this situation, it is important to know how many tests are correctly handled by the expert system without changing setting of winners (as defined in section 4 of chapter 9 ("a fast algorithm"). A formalization of the problem follows.

Problem name. Maximum set of compatible tests, decision version (MSCD).

Problem instance. A tree with alternating MIN and MAX boxes, multiplicative attenuators with rational values at the output of MIN boxes, bounded fan-in to MINs; a set of tests, TS; as assignment of ("expert-given") attenuations; an integer M.

Question. Is there a subset S of TS of size greater than or equal to M such that:

(a) all tests in S are satisfied;

(b) no switches occur at choice boxes, with respect to what indicated by the expert-given attenuations.

The Densest Hemisphere Problem (DHP) is solvable in polynomial time [Johnson and Preparata, 1978]. (Johnson and Preparata also show that the corresponding search problem is solvable in polynomial time.) DHP asks whether there is a subset of a given set of linear inequalities that has a solution and has cardinality larger than a given number. DHP is related to MSCD, because in the formalization given in section 4 of chapter 9 ("a fast algorithm") each test is a set of linear inequalities. However, in MCSD one has to consider subsets of tests, i.e., subsets of sets of linear inequalities, not simply subsets of linear inequalities. This leads to the conjecture that MCSD is NP-Complete and its associated search problem is NP-Hard.

Open problems: refinement with probabilistic sum

The section entitled "A fast algorithm" in chapter 9 presents a fast algorithm to solve the incomplete case for trees with MIN/MAX combinators and integrators when a condition on the winners at choice boxes holds. (See that section for details.) Now, it will be shown that the obvious generalization of that algorithm to the MIN/p+ case leads to a very slow algorithm.

Consider the incomplete test case in MIN/p+ trees. This is known to be a NP-Complete problem (or NP-Hard, depending on the CF alphabet and the choice of attenuations). A slow algorithm to solve this case is presented here.

Let an inference tree (suitably defined) and a set of tests be given.

Algorithm 10.1

1. Choose influential bundles for each test. (The definition of influential bundle is definition 9.1.) (This is analogous to choosing a setting of winners in the MIN/MAX case.) (Use the best heuristic, for example choose bundles that are "closest" to the ones used by setting attenuations to the expert-given ones. It must be recalled that the problem remains hard even if the correct influential bundles are known for each test--see the last theorem in the second section of chapter 9, "synthesis of attenuations when principal paths are known.")

2. The resulting constraints contain products of (unknown) attenuations; call them a_i, for different i's. Linearize them by setting each product to a new variable; call each new variable z_i, for a different i. (If the z's are known, the a's

can easily be obtained by back-substituting, or by taking the logarithms of both sides of each equation equating a product of a's to a z.)

3. To find **a** solution satisfying the tests, one needs to solve a linear system in the z's; to find <u>the closest</u> solution to the expert-given attenuations, one needs to solve a quadratic programming problem, if the Euclidean distance is used.

4. If no solution exists, go to 1. (And choose another influential bundle.)

(End of algorithm)

<u>Analysis</u>. The analysis concerns only the number of z_i's introduced at step 2, because the other aspects of the algorithm, notably the choice of influential bundles, are analogous for the MIN/MAX case. In particular, note that the solution method for quadratic programming problems in [Gottfried and Weisman, 1973, 214-215] can be followed without computational troubles if one assumes that no subtrees are irrelevant, because this is what a z_i being equal to zero translates to in the model.

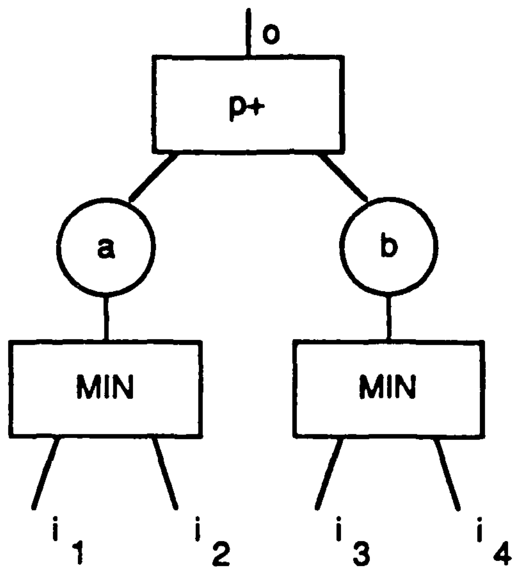
The number of distinct auxiliary variables ("z_i's") is exponential in the size of the tree, even for trees of small fixed depth. This makes the above algorithm impractical except for small problems.

(End of analysis)

Here is an example of use of the algorithm just given.

<u>Example 10.2</u>

Consider the tree, the tests, and the expert-given attenuations shown in Figure 10.2.



	i_1	i_2	i_3	i_4	o
T_1	.5	.4	.7	.4	.09
T_2	.9	.8	.8	.9	.36

$a^E = .2$, $b^E = .3$

Figure 10.2 Instance of the refinement problem with probabilistic sum

(Step 1) The choice of bundle is fixed, in this simple case. (It would not be fixed in general, e.g. for deeper trees.) The bundles are shown in Figure 10.3 (a) and (b) for tests T_1 and T_2, respectively.

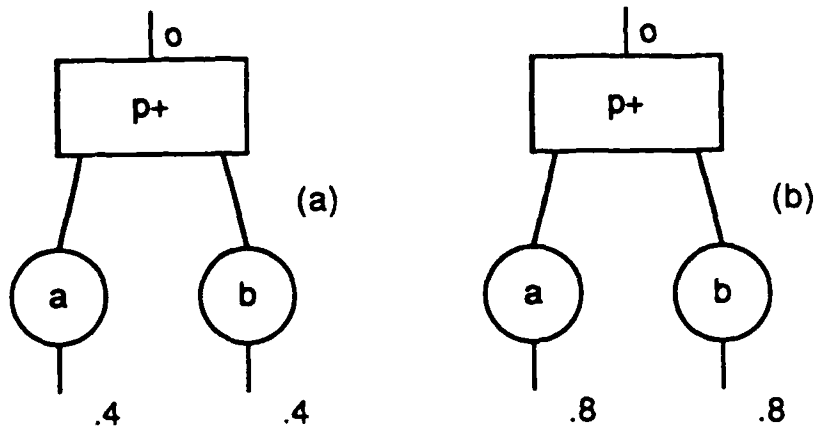


Figure 10.3 Influential bundles for the examples in Figure 10.2

(Step 2) The constraints are:

$.4a[p+].4b = .09$

$.8a[p+].8b = .36$

(The first constraints corresponds to test T_1, the second one to test T_2.) Or:

$.4a + .4b - .16ab = .09$

$.8a + .8b - .64ab = .36$

Let ab = z:

$.4a + .4b - .16z = .09$

$.8a + .8b - .64z = .36$

(Step 3) A solution to the system is a = b = .25, z = .0625. (Since there is a solution, in this simple example step 4 is never executed.) The quadratic programming problem is:

$.4a + .4b -.16z = .09$

$.8a + .8b - .64z = .36$

$\min y = (a-a^E)^2 + (b-b^E)^2 + (z-z^E)^2$, where $z^E=a^E b^E=.06$.

(End of example)

REFERENCES

Aho, A.V., J.E.Hopcroft, and J.D. Ullman. The Design and Analysis of Computer Algorithms. Reading, MA: Addison-Wesley, 1974.

Brachman, R.J. Quoted in Van de Riet, R.P. "Conference Report: Expert Database Systems." A report on the first conference on Expert Database Systems, appeared in Future Generation Computer Systems, 2, 3 (September 1986), 191-196.

Breuer, M.A. and A.D. Friedman. Diagnosis and Reliable Design of Computer Systems. Potomac, Maryland: Computer Science Press, 1976.

Brodie, M. (ed.). Transcript of discussions from a working group on knowledge base management systems, at the "First International Workshop on Expert Database Systems," Kiawah Island, South Carolina, October 24-27, 1984.

Brooks, R. and J. Heiser. "Some Experience on Transferring the MYCIN System to a New Domain." IEEE Transactions on Pattern Analysis and Machine Intelligence, 2, 5 (September 1980), 477-478.

Brownston, L., R. Farrell, E. Kant, and N. Martin. Programming Expert Systems in OPS5--An Introduction to Rule-Based Programming. Reading, Massachusetts: Addison-Wesley, 1985.

Cheeseman, P. "Learning of Expert Systems from Data." Proceedings of the IEEE Workshop on Principles of Expert Systems (1984), 115-122.

Chi, M.T.H., R. Glaser, and E. Rees. "Expertise in Problem Solving." Technical report No.5, Learning Research and Development Center, University of Pittsburgh, May 1981. Also in: Sternberg, R. (ed.). Advances in the Psychology of Human Intelligence. Hillsdale, New York: Erlbaum, 1981.

Clocksin, W.F. and C.S. Mellish. Programming in Prolog. Berlin: Springer Verlag, 1981.

Davis, R. and J. King. "An Overview of Production Systems."

In: Elcock, E.W. and D. Michie (eds.). Machine Intelligence 8, 300-332. New York: Wiley, 1977.

Even, S. and O. Goldreich. "The Minimum-Length Generator Sequence Problem is NP-Hard." Journal of Algorithms 2, 3 (September 1981), 311-313.

Feigenbaum, E.A. "The Art of Artificial Intelligence. I. Themes and Case Studies of Knowledge Engineering." Proceedings of the Fifth International Joint Conference on Artificial Intelligence (1977), 1014-1029.

Forsyth, R. and R. Rada. Machine Learning: Applications in Expert Systems and Information Retrieval. London: Ellis Horwood, 1986.

Fujiwara, H. and S. Toida. "The Complexity of Fault Detection Problems for Combinatorial Logic Circuits." IEEE Transactions on Computers, 31, 6 (June 1982), 555-560.

Gallant, S.I. "The Pocket Algorithm for Perceptron Learning." College of Computer Science Technical Report SG-85-19, Northeastern University, Boston, Massachusetts, January 2, 1985.

Garey, M.R. and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: Freeman, 1979.

Gashnig, J. "Prospector: An Expert System for Mineral Exploration." In Michie, D. Introductory Readings in Expert Systems. New York: Gordon and Breach Science Publishers, 1982.

Gold, M. "Language Identification in the Limit." Information and Control, 10, 447-474 (May 1967).

Gottfried, S.B. and J. Weisman. Introduction to Optimization Theory. Englewood Cliffs, New Jersey: Prentice-Hall, 1973.

Hafner, C.D. Contribution to "ACM Forum." Communications of the ACM, 29, 7 (July 1986), p.592.

Håjek, P. "Combining Functions for Certainty Factors in Consulting Systems." Preprint of a paper presented at the Second International Conference on Artificial Intelligence and Information--Control Systems of Robots, Smolenice, Czechoslovakia, 1982.

Hardy, G.H. and E.M. Wright. An Introduction to the Theory of Numbers, 5th ed. Oxford: Oxford University Press, 1979.

Hayes-Roth, F. "Knowledge-Based Expert Systems." Computer, 17,10 (October 1984), 263-273.

Hayes-Roth F., D.A. Waterman, and D.B. Lenat. Building Expert Systems. Reading, Massachusetts: Addison-Wesley, 1983.

Ibarra, O.H., and S.K. Sahni. "Polynomially Complete Fault Detection Problems." IEEE Transactions on Computers, 24 (1975), 242-249.

Johnson, D.S. "The NP-Completeness Column: An Ongoing Guide." Journal of Algorithms, 4, 1 (March 1983), 87-100.

Johnson, D.S. and F.P. Preparata. "The Densest Hemisphere Problem." Theoretical Computer Science, 6, 1 (February 1978), 93-107.

Khaciyan, L.G. "A Polynomial Algorithm in Linear Programming." Soviet Mathematics Doklady, 20 (1979), 191-194.

Kowalski, R. Logic for Problem Solving. New York: North Holland, 1979.

Kozen, D. "Lower Bounds for Natural Proof Systems." Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (1977), 254-266.

Kozlov, M.K., S.P.Tarasov, and L.G. Khaciyan. "Polynomial Solvability of Convex Quadratic Programming." Soviet Mathematics Doklady, 20 (1979), 5.

Kulish, U.V. and W.L. Miranker. Computer Arithmetic in Theory and Practice. New York: Academic Press, 1981.

Kulisch, U.W. and W.L. MIranker. A New Approach to Scientific Computation. New York: Academic Press, 1983.

Loveland, D.W. "Finding Critical Sets." Report CS-1982-23, Department of Computer Science, Duke University (1982).

Lu, H. and S.C. Lee. "Fault Detection in M-Logic Circuits Using the M-Difference." Proceedings of the 14th IEEE Multiple-Valued Logic Conference (1984), 62-70.

Megiddo, N. "Is Binary Enc[illegible]ing Appropriate for the Problem-Language Relationship?" Theoretical Compueter Science, 19 (1982), 337-341.

Megiddo, N. "Towards a Genuinely Polynomial Algorithm for Linear Programming." SIAM Journal on Computing, 12, 2 (May

1983), 347-353.

Michalski, R.S. and R.L. Chilauski. "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis." International Journal of Policy Analysis and Information Systems, 4 (1980), 2, 125-161.

Michalski, R.S. and R.L. Chilauski. "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples: A Case Study Involving Soybean Pathology." International Journal of Man-Machine Studies, 12 (1980), 63-87.

Minski, M. and S. Papert. Perceptrons: An Introduction to Computational Geometry. Cambridge, Massachusetts: MIT Press, 1969.

Nau, D. "Expert Computer Systems." Computer, 16, 2 (February 1983).

Nigrin, A. Term report for CPS215, Fall 1984, Department of Computer Science, Duke University, Durham, North Carolina.

Nilsson, N.J. Learning Machines. Foundations of Trainable Pattern-Classifying Systems. New York: McGraw-Hill, 1965.

Politakis, P.G. "Using Empirical Analysis to Refine Expert System Knowledge Bases." Report CBM-TR-130, Laboratory for Computer Science Research, Rutgers University, New Brunswick, New Jersey, October 1980.

Politakis, P.G. and S. Weiss. "Designing Consistent Knowledge Bases: An Approach to Expert Knowledge Acquisition." Report CBM-TR-113, Laboratory for Computer Science Research, Rutgers University, New Bruswick, New Jersey, March 1980.

Politakis, P.G. and S. Weiss. "Using Empirical Analysis to Refine Expert System Knowledge Bases." Artificial Intelligence, 22, 1 (January 1984), 23-48.

Prade, H. "A Synthetic View of Approximate Reasoning Techniques." Proceedings of the Eight Internationa Joint Conference on Artificial Intelligence (1983), 130-136.

Quinlan, J.R. "Inferno: A Cautious Approach to Uncertain Inference." Computer Journal, 26 (1983), 3, 255-269.

Quinlan, J.R. "Consistency and Plausible Reasoning." Proceedings of the Eight International Joint Conference on Artificial Intelligence (1983), 137-144.

Rada, R. "Probabilities and Predicates in Knowledge Refinement." _Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems_ (1984), 123-128.

Rada, R. "Gradualness Facilitates Knowledge Refinement." _IEEE Transactions on Pattern Analysis and Machine Intelligence_, 7, 5 (September 1985), 523-530.

Rada, R., S. Humphrey, N. Miller, C. Coccia, and M. Dominiak. "Knowledge Refinement for Information Retrieval." Typescript, National Library of Medicine, Bethesda, MD, USA, 1985.

Reboh, R. _Knowledge Engineering Techniques and Tools for Expert Systems_. Linkoping Studies in Science and Technology, no.71. Linkoping, Sweden: no publisher, 1981.

Richer, M.H. "An Evaluation of Expert System Development Tools." _Expert Systems_, 3, 3 (July 1986), 166-183.

Rosenbloom, P.S., J.E. Laird, J. McDermott, A. Newell, and E. Orciuch. "R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem Solving Architecture." _IEEE Transactions on Pattern Analysis and Machine Intelligence_, 7, 5 (September 1985), 561-569.

Ruspini, E.H. "Possibility Theory Approaches for Advanced Information Systems." _Computer_, 15, 9 (September 1982), 83-91.

Sahni, S. "Computationally Related Problems." _SIAM Journal on Computing_, 3, 4 (December 1974), 262-279.

Saitta, L., A. Giordana, A. Molli, and D. Timpanaro. "BIMBO: A System Which Learns Its Expertise." _Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems_ (1984), 99-106.

Shapiro, E.Y. "Inductive Inference of Theories from Facts." Technical Report 192, Yale University, Department of Computer Science (February 1981).

Shapiro, E.Y. "Algorithmic Program Debugging." Research Report 237, Yale University, Department of Computer Science (May 1982). (Also published as a book with the same title: Cambridge, Massachusetts: MIT Press, 1983.)

Shapiro, E.Y. "Logic Programs with Uncertainties: A Tool for Implementing Rule-Based Expert Systems." _Proceedings of the Eight International Joint Conference on Artificial Intelligence_ (1983), 529-532.

Shortliffe, E.H. Computer-Based Medical Consultations: MYCIN. New York: Elsevier, 1976.

Slagle, J.R., M.W. Gainor, and E.J. Halpern. "An Intelligent Control Strategy for Computer Consultation." IEEE Transactions on Pattern Analysis and Machine Intelligence, 6, 2 (March 1984), 129-136.

Stonebraker, 1984. Oral quotation from the discussion transcribed by Brodie [1984].

Valtorta, M., B.T. Smith, and D.W. Loveland. "The Graduate Course Advisor: A Multi-Phase Rule-Based Expert System." Proceedings of the IEEE Workshop on Principles of Expert Systems (1984), 53-57.

Van Emden, M.H. and R.A. Kowalski. "The Semantics of Predicate Logic as a Programming Language." Journal of the ACM, 23, 4 (October 1976), 733-742.

Weiss, S. and C.A. Kulikowski. "Expert: A System for Developing Consultation Models." Proceedings of the Sixth International Joint Conference on Artificial Intelligence (1979), 142-147.

BIOGRAPHY

Marco Valtorta was born in Milan, Italy, on May 7, 1956. He obtained the Laurea in Ingegneria Elettronica Degree (cum laude) from the Politecnico di Milano in July 1980, with a thesis on the computation of heuristics for the A* algorithm. He was awarded a Fulbright scholarship to pursue a Doctoral Degree in Computer Science at Duke University starting in August 1980. He obtained the Master of Arts degree in Computer Science in April 1983, developing a prototypical expert system to advise graduate students in the selection of their course sequences. Since October 1985, he has been fully employed with the Commission of the European Communities, where he works for ESPRIT (European Strategic Programme of Research and development in Information Technologies) as a project monitor in the area of knowledge engineering.

Publications

"A Result on the Computational Complexity of Heuristic Search Estimates for the A* Algorithm." _Information Sciences_, vol. 34 (1984), 47-59.

"The Graduate Course Advisor: A Multi-Phase Rule-Based Expert System." _Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems_, December 1984.

"Detecting Ambiguity: an Example in Knowledge Evaluation." (Second author, with D.W. Loveland.) _Proceedings of the Eighth International Joint Conference in Artificial Intelligence_, August 1983.

Marco Valtorta also wrote other conference papers (published in the proceedings of the conferences) and internal reports.

END
12-87
DTIC